REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1996	Final Apr. 1994 - Sep. 1995

4. TITLE AND SUBTITLE
Feasibility of Integrating IDEF Methodology with Testing of System Dynamics (Short Term Project-STP#24)

5. FUNDING NUMBERS
C-DLA900-88-D-0383
PE-7811s
PR-88003

6. AUTHOR(S)
T. O. Boucher and M. A. Jafari

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rutgers, The State University of New Jersey
The Center for Advanced Food Technology
Cook College, NJ Agricultural Experiment Station
New Brunswick, NJ 08903

8. PERFORMING ORGANIZATION REPORT NUMBER
FTR18.0

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Logistics Agency
8725 John J. Kingman Road
Ft. Belvoir, VA 22060-6221

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The "Integrated Computer Aided Manufacturing - Definition" (IDEF) methodology, developed by the U.S. Air Force to analyze manufacturing, lacks a mechanism for direct evaluation of system performance. Prototype software entitled "IDEF/System Dynamics" was developed which allows a user to define a manufacturing system as an IDEF0 model and automatically generate a simulation model of the system. After the user adds time durations of events and the initial state of the system, the simulation model can be run. The prototype software demonstrates the feasibility of the overall approach including the necessary algorithms and database interaction. The IDEF/System Dynamics approach to manufacturing system design can be used for either a "greenfield" project or to modernize the current operations of an existing plant. In order to reach a commercial software user audience, it will be necessary to enhance the prototype to allow models to be entered graphically (presently the model is entered in tabular form) and to add more controller functions such as counters, timers and data manipulation elements.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	80
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Contents

1.0 CRAMTD STP #24
Results and Accomplishments

This Final Technical Report covers the activities for CRAMTD Short Term Project, STP #24, "Feasibility of Integrating IDEF Methodology with Testing of System Dynamics".

1.1 Introduction and Background

The objective of this project was to test the feasibility of automatically generating a simulation model of a system designed using the IDEF methodology. During the conduct of STP #4 it was found that the IDEF methodology, which provides static models of the functional and informational architectures of a CIM system, lacked a mechanism for direct evaluation of system performance. In order to evaluate the dynamic performance of a system design, a designer would have to separately specify a simulation model (IDEF2). A methodology was defined in concept that could take the IDEF specification of a manufacturing system and automatically generate a model that can be used to analyze system performance (IDEF2). Furthermore, the basic methods were developed to also define the controller logic for operating the system and to automatically generate the control code. The final report of STP #4 recommended that a new STP be defined to demonstrate the feasibility of integrating the IDEF methodology with testing system dynamics. This became STP #24, definitized in April, 1994.

1.2 Results and Conclusions

A prototype software entitled *IDEF / System Dynamics* was developed based on a software requirements specification. The software was developed under Visual Basic and is user interactive in a Windows environment. The software allows a user to define a system as an IDEF0 model. The model is entered in tabular form instead of graphical form. Once entered, the user can automatically generate a simulation model of the system. After the user adds additional information, such as time durations of events and the initial state of the system, the simulation model can be run. The interactive screen is updated with the current state of the system as the simulation continues. We conclude from this that the feasibility of directly constructing the simulation model from IDEF0 has been demonstrated.

Additional modules were developed under *IDEF / System Dynamics* for the purpose of demonstrating interaction with a database and the generation of control programs. IDEF0 functional models usually specify the information flows that take place among related functions. In order to capture information flows in the simulation, an extension to IDEF0 was defined. That extension allowed an IDEF0 activity to be a predefined object that would interact with a database by providing predefined information. The interaction is between the *IDEF / System Dynamics* software and an Microsoft Access database file. When an output of an IDEF0 activity is information to be stored in a database, the predefined object executes an "insert" instruction, placing that information into the database file. Although this feature was implemented on a very

preliminary basis, we conclude that it demonstrates the feasibility of including data flows within the simulation of an IDEF0 model.

When the IDEF0 model is a representation of a set of machines under automatic control, the simulation model is actually showing the sequencing of events that occurs among machines. In that sense the simulation model contains the overall control logic for the sequencing of operations. Algorithms were implemented to take that sequencing logic and define a supervisory control program that could regulate those sequencing activities. Though limited to discrete inputs and outputs, the algorithm was implemented and illustrated the capability of generating such control logic in ladder logic format.

In order to prove the commercial viability of this software, it would be necessary to add more controller functions, such as counters, timers and data manipulation elements. This is recommended as further development in Section 1.3, Item 3.

The IDEF/Systems Dynamics approach to manufacturing system design can be used for either a "greenfield" project or to modernize the current operation of an existing plant. It is probably most appropriate for modernization projects because of the requirements for data in the simulation model. Data on existing operations could be used to define the "as is" IDEF model of the system. The simulation of this "as is" system should yield results consistent with the current operation, thus giving credibility to the "as is" simulation model. The "to be" model would be a perturbation on the "as is" model, showing the user the magnitude of the improvement in operation for each modernization change.

At the present time IDEF/System Dynamics takes deterministic time data as input. It is typical for production systems to generate events in stochastic time, i.e., a probabilistic description of event time is required. This is a fairly minor extension to the software, but should be implemented before it is used in a commercial project. This is recommended under Item 1 in Secion 1.3.

A User Manual was written and is appended to this final report.

1.3 Recommendations

Based upon the experience developing this prototype software, we recommend:

1. Further development be funded for enhancing the IDEF / System Dynamics prototype software into a more commercial software package.
2. Software be developed in C++ to allow a user to enter models graphically, as is the usual practice in IDEF modeling. Microsoft Visual Basic has limited graphics capability; for example, a user cannot scroll beyond a single screen, limiting the size of a model that can be entered graphically.
3. Extend the control logic generating capability of the software to include timing and counting functions, communication protocols, and data handling functions.

2.0 Program Management

This STP was proposed as a three phase activity as illustrated in Figure 1, "Time & Events and Milestones" (Appendix 4.1). These include the following:

Phase I Define Prototype Software Specification and Develop Prototype Algorithms

Phase II Implementation and Feasibility Demonstration

Phase III Provide Documentation

The key first phase objective is to develop algorithms to translate IDEF0 specifications into an intermediate model that can be analyzed. Computer code was to be developed for those algorithms in the second phase and implemented on an IBM PC class computer. The feasibility demonstration consisted of constructing test cases of different manufacturing control situations and running the prototype software against these test cases.

2.1 Summary of STP Accomplishments

- The feasibility of integrating IDEF0 with the automatic generation of a simulation model was demonstrated.
- The feasibility of integrating data flows to a database within the simulation model was demonstrated.
- Prototype software was built that would allow IDEF0 model building and simulation.
- A user manual was written for distribution with the prototype software.

3.0 Short Term Project Activities

3.1 STP Phase I Tasks

3.1.1 Define Prototype Software Specification and Develop Prototype Algorithms (Task 3.4)

This phase concerned preliminary work to be done in preparation for writing software code. It is necessary to develop a planning document that specifies the functionality of the software in sufficient detail to guide the programmer. It is also necessary to design some of the important algorithms or procedures that will be implemented.

3.1.1.1 Define Prototype Software Specification (3.4.1)

The work on developing a software requirements specification began in the third quarter of 1994. It was decided to write software that would run under Windows in an interactive mode to provide a user-friendly interface. Visual Basic 3.0 was chosen as the development language due to its ability to create a professional looking interface with a minimum of graphics programming effort. The scope of the effort was defined to include an IDEF0 development screen, an IDEF1X development screen, and two modules for model analysis: one for simulation and one for defining control code. The relationship between IDEF0 and IDEF1X was to be made explicit in the simulation by providing the ability to interact with a database file that could contain tables based on the IDEF1X data model. The software requirements specification was published as Technical Working Paper (TWP) #99, "IDEF/Systems Dynamics Evaluation Software: Software Requirements Specification, Version 1.0". It was provided as an appendix to the interim technical report (IRT 24.2) for the period ended February, 1995.

3.1.1.2 Algorithm Development (3.4.2)

Algorithm development began in the third quarter of 1994 and continued through the spring of 1995. The basis of the algorithms and procedures for converting an IDEF0 model to a simulation model in the modeling formalism called Petri nets was previously developed by the principal investigators on this project. These methods had to be refined during this phase of the project.

3.2 STP Phase II Tasks

3.2.1 Implementation and Feasibility Demonstration (Task 3.5)

This phase focused on writing computer code for the algorithms being developed in Phase I. It consisted of two tasks: software implementation and feasibility demonstration.

3.2.1.1 Software Implementation (3.5.1)

In January, 1995, we implemented the workstation for developing software. Visual Basic 3.0 and Microsoft Access were purchased and installed. We began with the design interface for IDEF0, which was intended to be implemented in graphics mode. However, we subsequently found that the limitations of screen size made graphics mode usable only for very small problems. A subsequent decision was made to provide data entry screens in matrix mode, which would allow a user to enter very large models without difficulty.

Simultaneously, we were designing the modules for Petri net simulation and interaction with the Access database. By May, 1995, we had linked the simulation model and the database.

During the spring and summer of 1995, we brought the two modules together by implementing the procedures that would generate the simulation model from the IDEF model. Initially we found that certain IDEF model structures resulted in simulation models that were non-cyclic; i.e., events of the simulation did not repeat themselves as in a closed system. This led to substantial revisions of the simulation software to allow non-cyclic system behavior to be simulated. It was during this period that we also implemented the module to generate the system control logic.

3.2.1.2 Feasibility Demonstration (3.5.2)

Manufacturing problem situations were developed to use for testing the functionality of the software. These tests were conducted satisfactorily during the late summer, 1995. Demonstration problems are provided in the User Manual, which is appended to this final report.

3.3 STP Phase III Tasks

3.3.1 Provide Documentation (3.6)

This phase consisted of reporting and documenting the results of the STP.

3.3.1.1 Develop Software User Manual (3.6.1)

A software user manual was developed that describes the functionality of the software. The manual was released as Technical Working Paper (TWP #110) and is attached to this final report.

4.0 Appendix

4.1 Figure 1 "Time & Events and Milestones"
4.2 Technical Working Paper (TWP) 99, "IDEF/Systems Dynamics Evaluation Software: Software Requirements Specification, Version 1.0"
4.3 Technical Working Paper (TWP) 110, "User Manual: IDEF/System Dynamics Software, Version 1.0a".

Figure 1 - CRAMTD Short Term Project #24
Integrated IDEF Methodology
Projected Time & Events and Milestones

Task Name	Reference	1994									1995								
		A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S
Phase I Define Prototype	3.4																		
Software Specification	3.4.1																		
Algorithm Development	3.4.2																		
Phase II Implementation/Demo	3.5																		
Software Implementation	3.5.1																		
Demonstration	3.5.2																		
Phase III Documentation	3.6																		
Software User Manual	3.6.1																		
Final Report	3.6.2																		

Printed: 6/25/96

COMBAT RATION ADVANCED MANUFACTURING TECHNOLOGY DEMONSTRATION (CRAMTD)

IDEF/Systems Dynamics Evaluation Software: Software Requirements Specification, Version 1.0

Technical Working Paper (TWP) 99

T.O. Boucher, M.A. Jafari and R. Wurl
Department of Industrial Engineering

May 1995

Sponsored by:
DEFENSE LOGISTICS AGENCY
Cameron Station
Alexandria, VA 22304-6145

Contractor:
Rutgers, The State University of New Jersey
THE CENTER FOR ADVANCED FOOD TECHNOLOGY*
Cook College
N.J. Agricultural Experiment Station
New Brunswick, New Jersey 08903

Dr. John F. Coburn
Program Director

TEL: 908-445-6132
FAX: 908-445-6145

*A New Jersey Commission on Science and Technology Center

1.0 General

1.1 The purpose of this software requirements specification is to comply with section 10.9 and task item 3.4.1 of STP #24 of Contract DLA 900-88-D-0383 between Rutgers University and the Defense Logistics Agency which requires that the contractor develop a specification for software that links an IDEF0 functional model and an IDEF1X informational model with a model that can analyze system dynamics. The contractor has reviewed methods for modeling system dynamics for discrete event systems that are compatable with the IDEF specification schema and has published prior work on the theory of how this can be accomplished. The contractor has also defined a methodology for generating controller programs for operating a discrete event system based on the IDEF functional specification. In order to test this methodology and to facilitate its use, the contractor proposes to develop user-friendly software. The purpose of this document is to provide the software requirements specification to guide that development effort. This specification is a working document and is subject to revision as the project proceeds.

1.2 Project References

There are two articles that summarize the theory on which this computer software will be based:

1. Boucher, T.O. and M.A. Jafari, "Design of a Factory Floor Sequence Controller from a High Level System Specification", *Journal of Manufacturing Systems*, Vol. 11, No. 6, 1992.

2. Jafari, M.A. and T.O. Boucher, "A Rule Based System for Generating a Ladder Logic Control Program from a High Level System Model", *Journal of Intelligent Manufacturing*, Vol. 5, No. 2, 1994.

Other related documentation as follows:

a) Project request: STP #24, Integration of IDEF and System Dynamics, March 1, 1994, Contract No. DLA 900-88-D-0383.

b) Previously developed technical documation relating to this project: See references (1) and (2), above.

c) Significant correspondence related to this project: None.

d) Documentation concerning related projects:

FTR6.0 Final Technical Report for STP#4 "Design and Development of a CIM Architecture for Food Manufacturing", T.O. Boucher and M.A. Jafari.

TWP52 "Information Architecture for Packaged Food Manufacturing", N.R. Adam, T.O. Boucher, T. Chamberlin, and J. Weber.

TWP37 "Functional Architecture for Packaged Food Manufacturing", T.O. Boucher, M.A. Jafari, S. Kim, and J. McPhail.

e) Manuals and documents that contain or explain technical factors affecting project development: None

f) Standards on reference documentation:

IDEF0 Functional Modeling Manual, ICAM Project Report, Contract #F33612-78-C-5158, Softech, Inc.,January, 1981

Information Modeling Manual IDEF1 - Extended, ICAM Project Report, Contract #F33615-80-C-5155, D. Appleton Company, Inc., December, 1985.

1.3 Terms and Abbreviations

1.3.1 Integrated Computer Aided Manufacturing - Definition (IDEF) Methodology: An approach to analyzing manufacturing developed by the U.S. Air Force which is now in the public domain.

1.3.2 IDEF0 is used to produce functional models, which are structured representations of the functions of a manufacturing system and the information and objects that interrelate those functions.

1.3.3 IDEF1X is used to produce informational models, which represent the

structure of information, or logical data base, needed to support functions of the manufacturing system.

1.3.4 IDEF2 is used to produce dynamic models, which are simulations of the time varying behavior of functions, information, and resources of a manufacturing system.

1.3.5 Petri nets are bipartite graphs which are used to model discrete event systems and to analyze those systems.

2.0 System Summary

2.1 Background

During the conduct of STP #4 it was found that the IDEF methodology, which provides static models of the functional and informational architecture of a CIM system, lacked a mechanism for direct evaluation of system performance. In order to evaluate the dynamic performance of a system design, a designer would have to separately specify a simulation model (IDEF2). Furthermore, since IDEF methodology can be used to define the specification of the manufacturing operations on the shop floor, it should be possible to derive the specification of controllers and their information flows directly from the IDEF models. This capability would enhance system specification, maintainability, and transfer to industrial users. A methodology has been defined in concept that could take the IDEF0 specification of the manufacturing system, automatically generate a dynamic model that can be used to analyze the system performance (IDEF2), and then automatically generate the computer code for a discrete controller to run the system. The final report of STP#4 recommended that a new STP be defined to demonstrate the feasibility of integrating the IDEF methodology with testing system dynamics. This became STP#24, definitized in April, 1994.

2.2 Objectives

There are six objectives to be satisfied by the proposed software. These objectives are as follows:

1. Design and build a software module to enter a functional architecture in IDEF0 format.
2. Design and build a software module and data base to enter an informational architecture in IDEF1X format.
3. Design and build a software module to convert an IDEF0 functional architecture to a dynamic model in Petri net format that will maintain the features of the functional architecture.
4. Design and build a dynamic link between the Petri net model and the IDEF1X data base.
5. Design and build the modules necessary to perform a simulation of the Petri net representation of the functional architecture, including interaction with the IDEF1X informational architecture data base.
6. Design and build the modules necessary to convert the Petri net specification of the functional architecture to code suitable for a discrete controller.

The software shall provide the above general specifications with a user friendly interface and the ability to document and save user sessions. System definition and functional diagrams follow in sections 2.3 and 2.4.

2.3 System Definitions

The following descriptions are referenced to figure 1, section 2.4, which is the architecture for the proposed software.

2.3.1 Project Selection: A main menu that allows the user to enter and exit the program. Program files are organized in directories by project. Once a project name is selected, entry choices shall include the EDITOR mode, the ANALYSIS mode, and the DATA DICTIONARY.

2.3.2 EDITOR Module: The EDITOR is used for entering, retrieving, changing, saving and deleting data. These functions are performed in interaction with a model and data representation Interface module. There is one for IDEF0 and one for IDEF1X.

2.3.3 IDEF0 Model and Data Representation Interface: A mouse driven IDEF0 menu and drawing template will be used to define Activities and Arcs. Arcs will include inputs, outputs, controls, mechanisms, signals and communications. The latter two concepts are extensions of the IDEF0 standard that will be required to completely specify a CIM application. The user defined IDEF0 drawing will automatically generate a data representation in matrix format. This will be used to save the model, redraw the model and provide the basis for converting the model to IDEF2 format.

2.3.4 IDEF1X Model and Data Representation Interface: A mouse driven IDEF1X menu and drawing template will be used to define Entities, Attributes and Relationships to define a logical database. These definitions will automatically generate database tables with the entity name and appropriate attribute fields. This interface will also be used for entering instances (records) of entities in their table data fields.

2.3.5 IDEF1X Database: This is the physical implementation of the IDEF1X model as generated from the IDEF1X Model and Data Representation Interface. It will be implemented on the Microsoft Access database management system, which will be linked to the IDEF1X Model and Data Representation Interface using Dynamic Data Exchange (DDE).

2.3.6 New File: This allows the user to open a new file as either an IDEF0 or IDEF1X file and give it a file name. Files reside in the current project directory. Once the file is established, the user is put into interaction with the appropriate IDEF Model and Data Representation Interface.

2.3.7 Open File: This allows the user to open an existing file. Once the existing file is requested by the user, the user is put into interaction with the appropriate IDEF Model and Data Representation Interface and the file is loaded.

2.3.8 Save File: Allows the user to save the current IDEF0 or IDEF1X file to disk.

2.3.9 Delete File: Allows the user to delete the current IDEF0 or IDEF1X file from disk.

2.3.10 Analysis Module: Must be entered by the user before any problem data can be executed. The analysis module allows the user to 1) convert the IDEF Models and Data Representation to a dynamic model representation, 2) simulate the dynamic model, 3) generate discrete event system controller code from the dynamic model, 4) print output and print results to files and 5) save current analysis sessions to memory and delete previous analysis sessions from memory.

2.3.11 IDEF2.Mak: Converts the IDEF Models and Data Representation to a dynamic model in Petri net representation. Retains the incidence matrix representation of the Petri net that is used to save the model and redraw the model.

2.3.12 IDEF2.Sim: Queries the user for dynamic model event times. Performs the simulation of the IDEF2 model, including interaction with the IDEF1X database. Retains simulation results in files.

2.3.13 Control.Mak: Queries the user to define physical input and output channels. Converts IDEF2 model to a generic code that can be applied to a programmable logic controller using discrete inputs and outputs.

2.3.14 Reports: Prepares output displays of 1) IDEF0 Model and Data Representation, 2) IDEF1X Model and Data Representation, 3) IDEF2 Model and Data Representation, 4) simulation results, 5) controller code model and data representation, and 6) Data Dictionary definitions. Also generates hard copy of above.

2.3.15 Save/Delete Session Module: Allows user to save all data and results of an analysis session in a file. Each session file will be uniquely named by the user. Allows the user to delete previously saved session files.

2.3.16 Current Problem Files: Contains data files currently in use as well as intermediate and final computations from problems in temporary storage. Data and results are retained only when requested through "Save/Delete Session" Module.

2.3.17 Data Dictionary: Contains user defined definitions of data element names used in IDEF models. This includes names of activities, arcs, entities and attributes.

2.4 Software Architecture: Referenced to Section 2.3

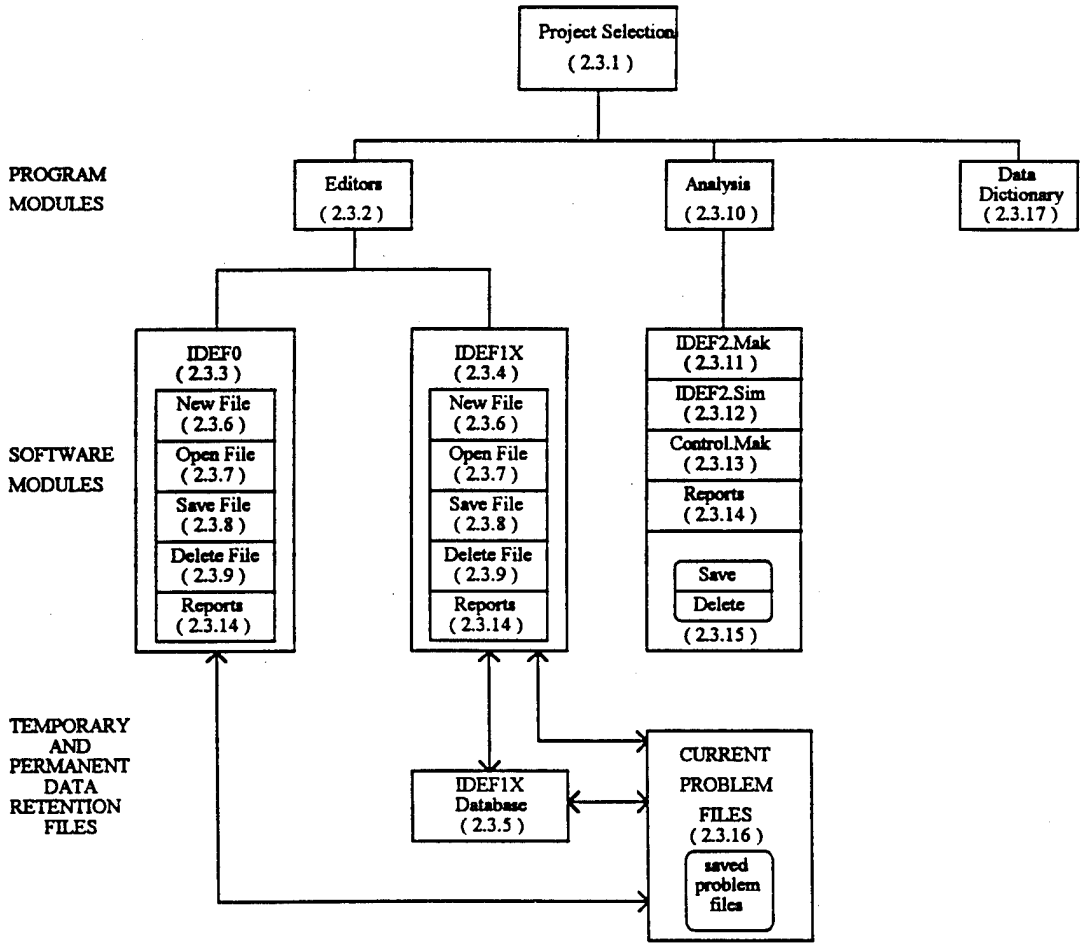


Figure 1 Architecture of Proposed IDEF/System Dynamics Software

2.5 Computer Program Identification

1. Editor Module
2. Analysis Module

2.6 Assumptions and Constraints

None identified.

3.0 Environment

3.1 Equipment Environment

a) PC, Windows Operating System, 4 meg RAM.

b) Storage Media 3.5" Floppy disk.

c) Output devices VGA monitor, printer.

d) Input devices Keypad, mouse.

e) No additional communications requirement.

3.2 Support Software Environment

End product software will be compiled to run on a Windows operating system.

Source code will be developed in Visual Basic 3.0.

3.3 Interfaces

None.

3.4 Security and Privacy

No requirement for security and privacy. Software will be public domain.

4.0 Detailed Characteristics and Requirements

4.1 Specific Performance Requirements

4.1.1 Accuracy and Validity

a) Mathematical calculations shall yield final results accurate to one decimal place.

b) Input data is subject to validity checks, the results of which are

presented to the user. User is responsible for insuring accuracy of final input data.

c) No data transmission checks required.

4.1.2 Timing

Timing is not crucial in the use of this software. Time duration for computations will be in the order of seconds.

4.2 Computer Program Functions

4.2.1 Identification of Computer Program No. 1

Program Name: Editor Module

This program is used for entering, modifying, deleting, saving and retrieving of problem data. The program modules and reference paragraphs are as follows:

a) IDEF0 Model and Data Representation Interface (2.3.3)

b) IDEF1X Model and Data Representation Interface (2.3.4)

c) New File Module (2.3.6)

d) Open File Module (2.3.7)

e) Save File Module (2.3.8)

f) Delete File Module (2.3.9)

g) Reports Module (2.3.14)

4.2.2 Identification of Computer Program No. 2

Program Name: Analysis Module

This program performs conversion of input data into a dynamic model, simulates the dynamic model, and provides code for programming a controller. It provides a facility for output of results and saving and deleting work. The program modules and reference paragraphs are as follows:

a) IDEF2.Mak (2.3.11)

b) IDEF2.Sim (2.3.12)

c) Control.Mak (2.3.13)

d) Reports Generator (2.3.14)

e) Save/Delete Sessions (2.3.15)

4.3 Inputs - Outputs

The following paragraphs describe the inputs and outputs of each program module identified in section 4.2.

IDEF0 Model and Data Representation (2.3.3)

Inputs - Activities, Arcs and their relationships in IDEF0 format

User changes to model structure

Names associated with Activities and Arcs

Outputs - Problem specific IDEF0 model

Matrices retaining the information contained in the IDEF0 structure

Files retaining the names associated with Activities and Arcs

IDEF1X Model and Data Representation (2.3.4)

Inputs - Entities, Attributes, their names and their data structures

Relationships between Entities

Primary Keys and Foreign Keys

User changes to model structure

Outputs - Problem specific IDEF1X model

Tables retaining the information contained in the IDEF1X model

Files retaining the IDEF1X model

IDEF1X Database (2.3.5)

Inputs - IDEF1X Model

Outputs - Data tables in Microsoft Access Database

New File Module (2.3.6)

Inputs - User request to establish new IDEF file

New file name

Outputs - User filename associated with current IDEF0 or IDEF1X window

Open File Module (2.3.7)

Inputs - File name of existing IDEF0 or IDEF1X file

Outputs - Retrieved IDEF0 or IDRF1X data file

Save File Module (2.3.8)

Inputs - User specified IDEF file name

Outputs - Saved IDEF0 or IDEF1X data file

Delete File Module (2.3.9)

Inputs - User specified IDEF file name

Outputs - Deleted IDEF data file

IDEF2.Mak Module (2.3.11)

Inputs - IDEF0 Data Representation

Outputs - Petri net graphical representation

Petri net incidence matrix

IDEF2.Sim Module (2.3.12)

Inputs - Petri net incidence matrix

Event timing

Outputs - Various performance results, such as reachability set, throughput, system deadlock analysis, average activity utilizations, and system cycle time.

Control.Mak Module (2.3.13)

Inputs - Petri net incidence matrix

Physical input and output channels

Outputs - Controller program

Reports Module (2.3.14)

Inputs - IDEF0 Input Data Files

IDEF1X Input Data Files

IDEF2 Output Data Files

IDEF2 Simulation Data Files

Controller Code Data Files

Outputs - Screen display of above files

Printer copy of above files

Save/Delete Session File (2.3.15)

Inputs - User specified file name

Outputs - Storage/Removal of complete session files

4.4 Data Characteristics

All input data will be standard integer. Resulting computations will be single precision numbers displayed in decimal format. The following are extimated storage requirements for the system (in bytes)

Programs	200,000
Databases	500,000
Data Dictionary	20,000
Total Storage	720,000 bytes

4.5 Failure Contingencies

This software is not critical to other system operations and will not have redundancy or fail safe provisions. Failure during operation will result in the loss of files not saved. Failure will require a restart and lost files will have to be reloaded.

4.6 Design Requirements

The only design requirement is modularity of functions. Each function will be designed to function independently. Parameter passing will link execution among

functions. Main program construction will be designed as calls to subroutines and functions.

4.7 Computer Security Requirements

None.

4.8 Human Performance Requirements

Human interface will be menu driven. No special human performance requirements.

5.0 Test and Qualification Requirements

Software test will be conducted by generating sample problems that will be solved manually. Comparison simulations of test problems will be run on a commercial simulation software and results will be compared to IDEF2.Sim output. These test problems will be used to verify intermediate and final computational results.

6.0 Notes

The following documents are cited in this specification and are available to the reader to assist in understanding this specification.

References:

1. Adam, N.R., T.O. Boucher, T. Chamberlin and J. Weber, "Informational Architecture for Packaged Food Manufacturing", TWP#52.

2. Boucher, T.O. and M.A. Jafari, "Design of a Factory Floor Sequence Controller from a High Level System Specification", *Journal of Manufacturing Systems*, Vol. 11, No. 6, 1992.

3. Boucher, T.O. and M.A. Jafari, Final Technical Report for STP#4 "Design and Development of a CIM Architecture for Food Manufacturing", FTR6.0.

4. Boucher, T.O., M.A. Jafari, S. Kim and J. McPhail, "Functional Architecture for Packaged Food Manufacturing", TWP#37.

5. Jafari, M.A. and T.O. Boucher, "A Rule Based System for Generating a Ladder Logic Control Program from a High Level System Model", *Journal of Intelligent Manufacturing*, Vol. 5, No. 2, 1994.

COMBAT RATION ADVANCED MANUFACTURING TECHNOLOGY DEMONSTRATION (CRAMTD)

IDEF/Systems Dynamics Software: User Manual, Version 1.0

Technical Working Paper (TWP) 110

T.O. Boucher, M.A. Jafari, R.C. Wurl, W. Zhao and G. Alpan
Department of Industrial Engineering

June 1996

Sponsored by:
DEFENSE LOGISTICS AGENCY
8725 John J. Kingman Rd.
Fort Belvoir, VA 22060-6221

Contractor:
Rutgers, The State University of New Jersey
THE CENTER FOR ADVANCED FOOD TECHNOLOGY*
Cook College
N.J. Agricultural Experiment Station
New Brunswick, New Jersey 08903

Dr. John F. Coburn
Program Director

TEL: 908-445-6132
FAX: 908-445-6145

*A New Jersey Commission on Science and Technology Center

1.0 INTRODUCTION

This manual describes software developed to enable a user to automatically generate a simulation model from an IDEF0 functional model that describes the functional operation of a production/manufacturing system. The user must first define the IDEF0 structure of the system. This structure is entered in a tabular format as opposed to an IDEF0 graphical format. Using a menu command, the IDEF0 model is converted to a simulation model. This is done using transformation rules that map the IDEF0 data structure into a Petri net incidence matrix. The Petri net serves as the underlying modeling structure for the simulation. For a description of Petri nets and their relation to IDEF0, the reader is referred to the bibliography, particularly references [1,2,4].

To run the simulation, the user enters a simulation analysis screen. Here the user is allowed to enter timing information concerning the length of time it takes for various activities to execute. Once the timing information is entered, the simulation can begin. The state of the system is updated on the simulation screen as the simulation executes.

In the following sections the user will be guided in the installation of the software, will be taken through the various screens, and will be guided through the execution of tutorials that illustrate typical manufacturing situations in the discrete parts and the batch process industries.

By clicking on "Project" with the mouse, a pull down menu appears that gives the following selections: "New Project", "Open Project", "Save Project", "Save Project As", and "Exit". Clicking on either "New Project" or "Open Project" will open a dialog box for the user to either establish a new project or open an existing project.

When the user selects "New Project", the dialog box shown in Figure 2 appears. Here the user clicks on the IDEF0 label and an input box appears to request the name of the new project.

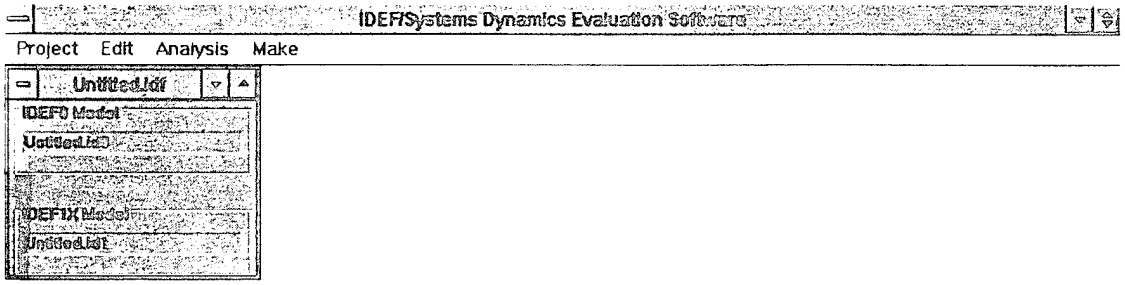


Figure 2. "New Project" Dialog Box

The "Open Project" dialog box is shown in Figure 3. Here the user can select an existing project file by clicking on the project filename. Project filenames have the extension ".idf".

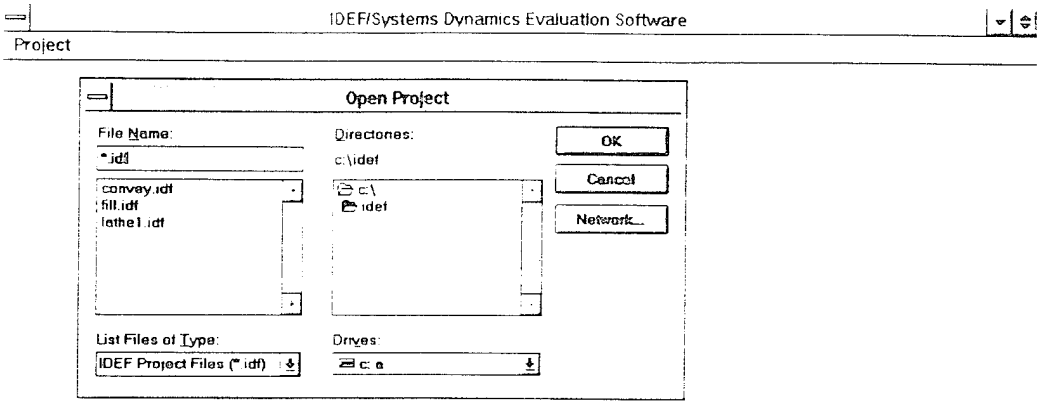


Figure 3. "Open Project" Dialog Box

The "Save Project" and "Save Project As" menu choices are not active until a project has been opened or a new project has been established. The "Exit" menu selection will take the user out of the IDEF program and put the user into the Windows Program Manager screen.

When a project is selected, the name of the project will appear on the screen as shown in Figure 4, below. In this figure we have opened the project "CONVEY.IDF". In addition, the user will be presented with three additional choices on the menu bar: "Edit", "Analysis", and "Make". The "Edit" option allows the user to enter data for a new model or to change data in an existing model. The "Analysis" option allows the user to create the simulation model and to perform the system dynamics analysis. The "Make" option allows the user to create a discrete event control program of the current model. These features will be discussed later.

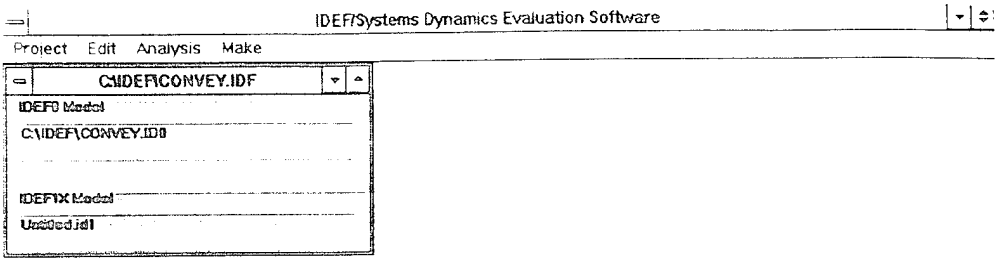


Figure 4. User Screen After Loading an IDEF0 Model

3.2 Using the Editor

When the user clicks on the "Edit" menu, a pull down menu offers three options. They are: "IDEF0", "IDEF1X", and "Input/Output". The "IDEF0" option puts the user into the IDEF0 model files for editing purposes. The "IDEF1X" option puts the user into the IDEF1X model files for creating a data table. The "Input/Output" option is specific to situations in which the operation of the system being modeled is governed by a controller. An example of this is an IDEF model of an automated machining cell or an automated packaging line which is controlled by a programmable controller. The "Input/Output" option allows the user to enter the signaling requirements of the controller in terms of input and output signals that will be used to control the system. A discussion and example of this will be shown later. Here we will focus on the creation of an IDEF0 model and simulating its system dynamics.

Clicking on the "IDEF0" option under the "Edit" menu brings up the IDEF0 input screen. If it is an existing project, the screen will show the most current input values that were previously entered. If it is a "New Project", the screen will be void of existing values, as shown in Figure 5. This is the screen into which all IDEF0 information is entered. The title bar at the top of the screen indicates the title of the software and the title of the project. Here we have established a new project called "TEST". There are six data entry areas provided for the user as follows: "Incidence Matrix", "Activities", "Arcs", "Mechanisms", "Intelligent Signals", and "Dumb Signals". In order to understand the meaning of these catagories and their relationship to standard IDEF concepts, we introduce the diagram in Figure 6, below.

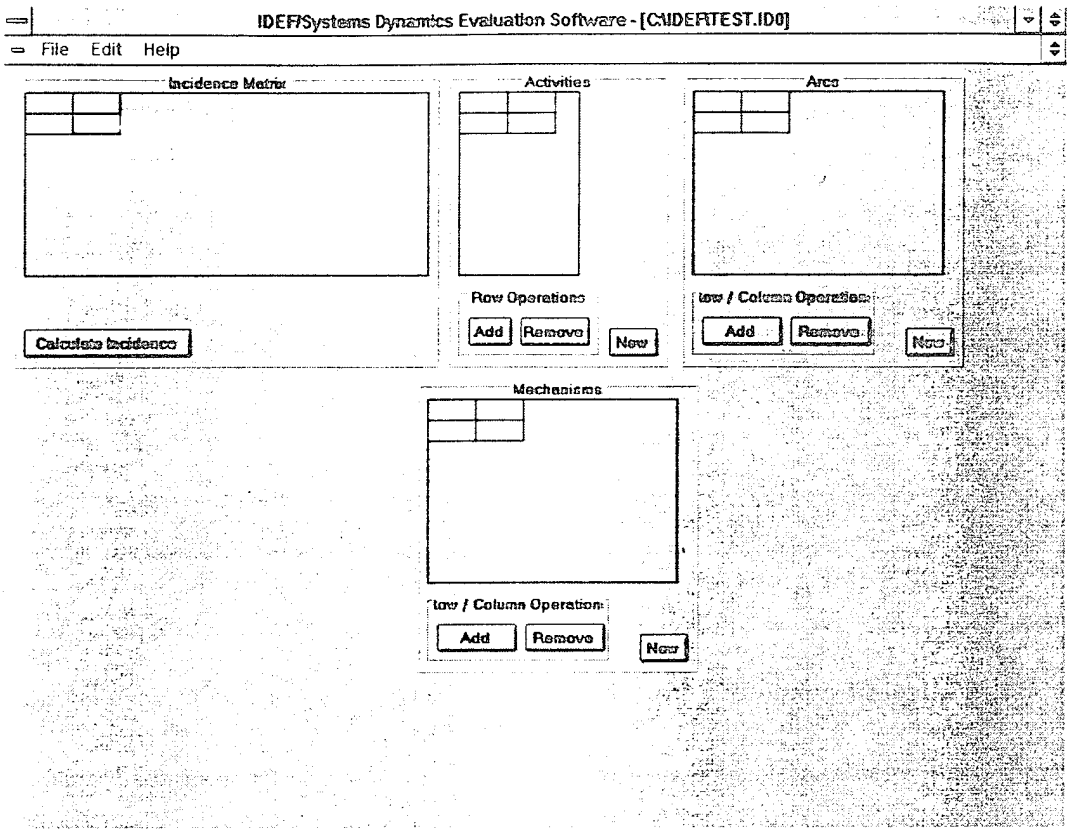


Figure 5. IDEF0 Model Input Data Screen

The building blocks for an IDEF0 model are the *activity box*, *input arcs*, *output arcs*, *mechanisms* and *controls*. The *activity box* defines a specific activity, or function, that is being modeled. The activity may be a decision-making or information-converting activity or a material converting activity. *Inputs* to the activity are shown as arcs entering from the left of the activity box. Inputs are items (material, information) that are transformed by the activity. *Outputs* of the activity are shown as arcs exiting at the right of the activity box. Outputs are a result of the activity acting on the inputs. *Controls* are shown as arcs entering the activity box from the top. A control is a condition that governs the performance of the activity. For example, a control may be a set of rules governing the activity or a condition that must exist before the activity can be done. *Mechanisms* are shown as arcs entering the activity box from the bottom. A mechanism is the means by which an activity is realized. For example, the mechanism may be a machine or a worker.

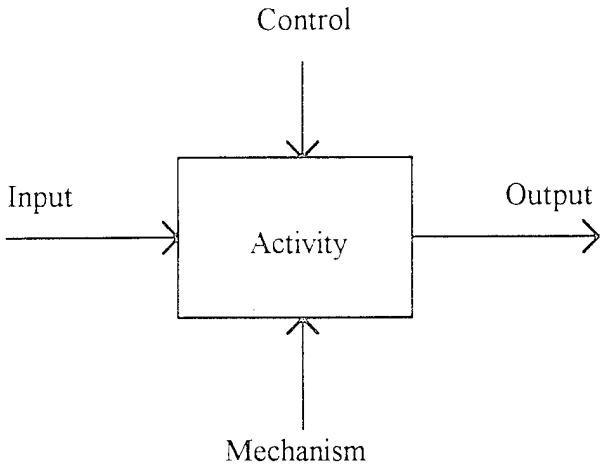


Figure 6. Building Blocks of IDEF0 Modeling

The activity box and the four entities of Figure 6 provide a concise expression: An *input* is transformed into an *output* by an *activity* performed by a *mechanism* and governed by a *control*. The specific activity, its inputs, outputs, mechanisms and controls must be defined for the situation being modeled. Activity boxes represent actions being performed and are labeled with verb phrases. Inputs, outputs, controls, and mechanisms are things, and are labeled with noun phrases. For a more thorough discussion of the IDEF0 modeling constructs, the reader is referred to reference [6].

A sequence of related activities can be described by an IDEF0 diagram, such as that shown in Figure 7. Here, two activities are related in sequence. The output of activity 1 becomes the input to activity 2. Any number of activities can be related to each other in order to describe a complete system. We will use Figure 7 as an illustration in this section and introduce more complex situations in Section 4, the tutorial section. We have adopted the following default labeling convention for activities(p), arcs(a), mechanisms (or resources) (r), and controls(s).

In order to illustrate the input conventions used with the screen of Figure 5, we will refer to the simple IDEF0 diagram of Figure 7. The input data that represents that diagram will be shown in Figure 8. The following describes the process by which the model of Figure 7 is input as data in Figure 8. The user begins with the "Activities" input box, Which is located in the upper center of Figure 8. By clicking on the "New" command button, a dialog box appears to ask for the number of activities. When the user responds, the activities are labeled consecutively from p1 to pn and appear in the activities box as shown. The user can "Add" and "Remove" from the activities list at any time by using the "Add" and "Remove" command buttons in the activities box. For the IDEF0 model of Figure 7, there are 2 activities. To the right of the activity is the "capacity" row. This is used to place a bound on the number of inputs that can utilize an activity simultaneously. For example, if the activity is a machining operation and the

machine tool is capable of machining only one part at a time, it is necessary to enter a "1" in the capacity cell next to the activity. If the activity is a transportation activity and the transport device is capable of holding only 2 units at a time, a "2" is entered in the capacity cell next to the activity. The user clicks on the left mouse button to select the cell. After the cell is selected, the user may enter a number from the keyboard or click on the right mouse button to increment the value in the cell by one. The user may click on the left mouse button to decrement the value in the cell by one.

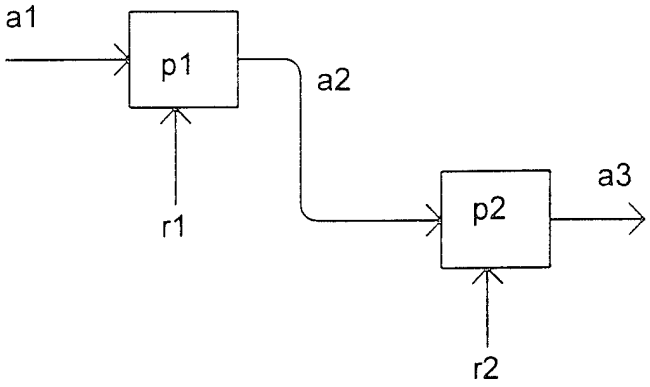


Figure 7 "TEST" IDEF0 Model

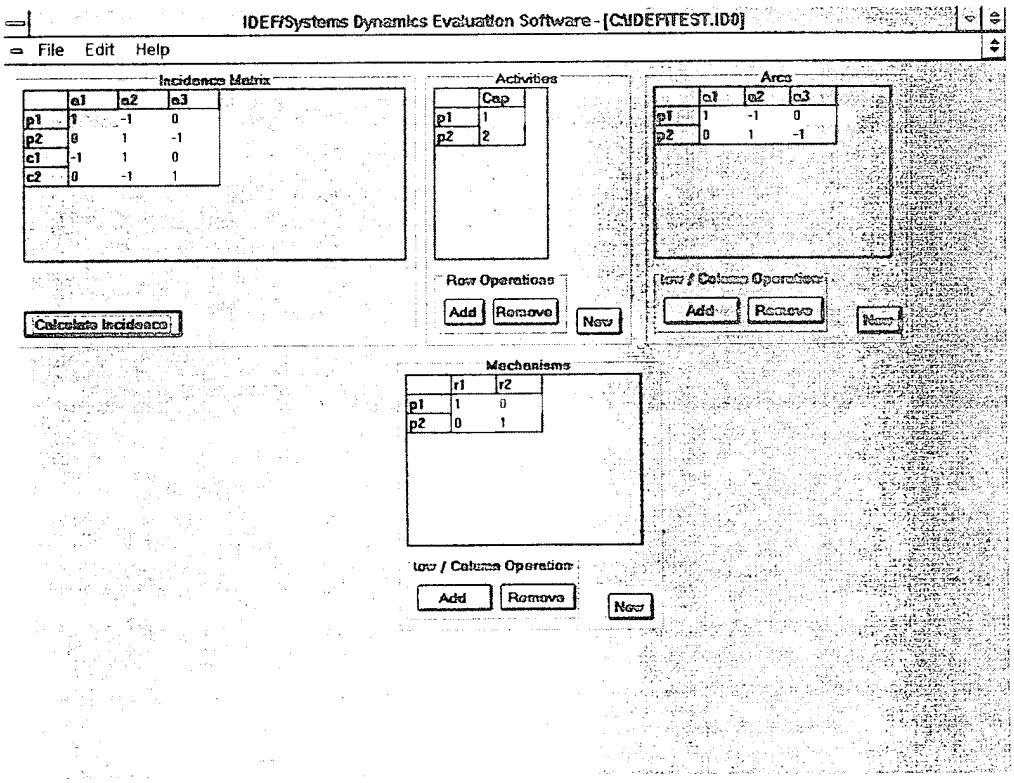


Figure 8 Input for "TEST" IDEF0 Model

The user then moves to the "Arcs" input box, shown at the upper right of Figure 8. In a manner similar to entering the number of activities, the number of arcs are input. In the IDEF0 model of Figure 7 there are 3 arcs. Arc labels from a1 to am are automatically provided. The user must then enter the relationship between activities and arcs in the arcs input box. If an arc is an *input* to an activity, the user must put a "1" in the cell representing that arc/activity combination. If an arc is an *output* of an activity, the user must put a "-1" in the cell representing that arc/activity combination. If there is no input/output relationship between an arc and an activity, the user leaves the cell blank, which is a value "0". In order to enter a value in the cell, the user first clicks on the cell with the left mouse button. This gives the user access to the cell. If the user clicks the left mouse button, the value in the cell decrements by one. If the user clicks the right mouse button, the value in the cell is incremented by one.

The user next moves to the "Mechanisms" input box to input the number of mechanisms in the same manner as the activities and arcs input box. In the case of our example problem, there are 2 mechanisms. If a mechanism services an activity, the user puts a "1" in the cell that shows the relationship between a particular mechanism and activity.

The controls structure of the IDEF0 model is unnecessary for the simulation model, which is used to show the movement of input and output flows as well as the utilization of resources. The control, or signal structure, is appropriate to the development of the system controller. This will be discussed later in Section 5.

The user now moves to the incidence matrix, which shows the relationship between the entities of the IDEF0 model. When the user clicks on the "Calculate Incidence" command button, the arc labels appear across the top of the box and the activities labels and the mechanisms labels appear down the rows of the box. The cells are automatically filled with data that is derived from the information previously entered by the user. This "incidence matrix" is a data structure representation of the complete IDEF0 model.

The user will note an additional symbol in the incidence matrix. The symbol "c" is a capacity holder which is derived from the information entered by the user in the "Activities" box. If the user declares that an activity will have a capacity associated with it, a capacity holder will be assigned the the activity. It's purpose will be apparent when we discuss the simulation model.

This is the data model from which the system dynamics model is created. At this point the user can save the model from the "Files" menu and can exit the IDEF0 editor.

3.3 Using the Simulator

Exiting the editor puts the user back into the main menu, shown previously in Figure 4. To enter the simulator, the user clicks on the "Analysis" menu item and selects "IDEF2-Simulation". This brings up the simulation screen, which is shown in Figure 9, below. The simulation screen has several input boxes and information displays as follows: "Incidence Matrix", "Initial Marking", "Previous/Current Marking", "Run Control", "Timing Vector", "Database Insert", and "Firing Vector".

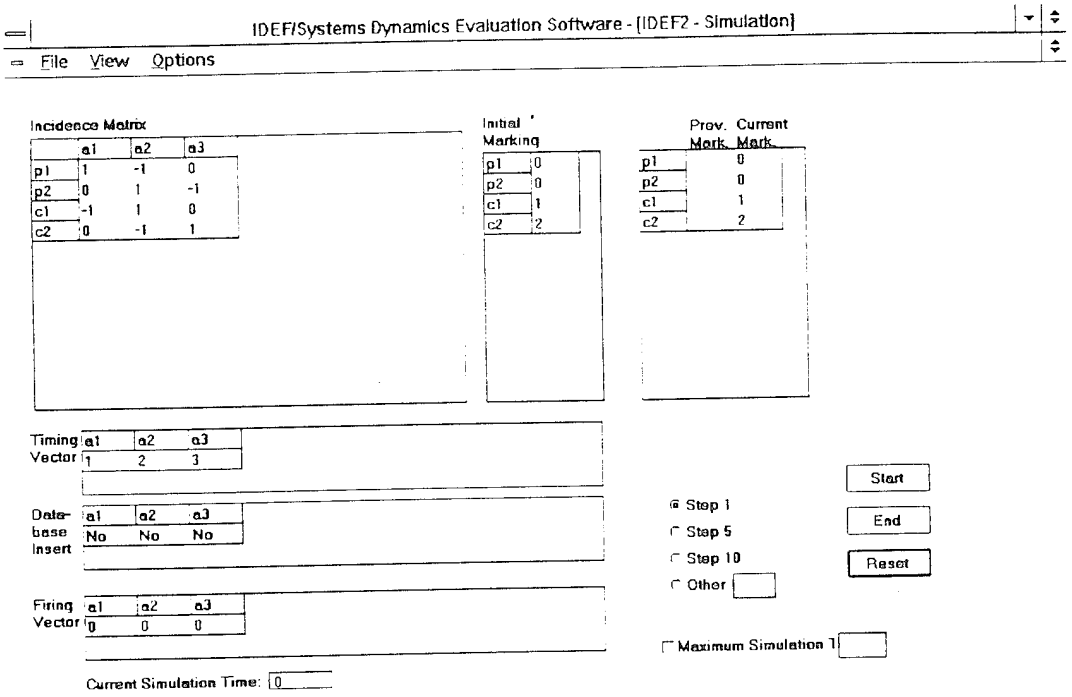


Figure 9 IDEF2 Simulation Screen

The "Incidence Matrix" is the data structure of the simulation model. It is derived from the "Incidence Matrix" (data structure) of the IDEF0 model of Figure 8, but it is not necessarily the same. For the "TEST" example, the IDEF0 incidence matrix and the simulation incidence matrix are identical. The simulation model uses the modeling formalism known as Petri nets. A Petri net is a bipartite graph having two types of nodes, known as "places" and "transitions". These nodes are connected by arcs. A graphical model of a Petri net uses circles to represent places and bars to represent transitions. When the IDEF0 model of Figure 7 is converted to a Petri net, the resulting model structure is as shown in Figure 10. Here you can see that the activities of the IDEF0 model have become the places of the Petri net and the arcs of the IDEF0 model have

become transitions of the Petri net. In the incidence matrix of the simulation model, the input places of a transition are labeled with a "-1" and the output places of a transition are labeled with a "1".

The capacity holder for an activity is shown as a complementary place across the place that represents the activity. The significance of this will be illustrated shortly.

Figure 10 is for illustration in this document. It is meant to clarify how the data structure is related to an underlying Petri net model. Graphical displays are not available in this software.

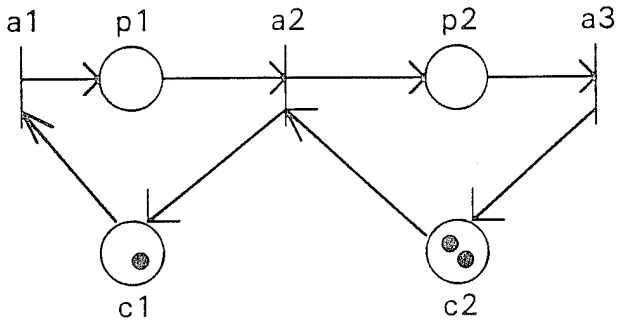


Figure 10 Petri net Representation of "TEST"

Before a simulation can be executed, it is necessary to initialize the model. The model is initialized by marking the active places. A place is marked by putting a token in it. For example, In Figure 10 there are two capacity places, c1 and c2. The available capacity is indicated by marking these places with tokens that equal their capacities. Presumably these capacities must be present for the system to be initialized. In the "Initial Marking" input box of Figure 9, we place a "1" next to c1 and a "2" next to c2.

Before a simulation can execute, it is necessary to provide timing information on how long it will take to perform an activity. There are two ways to do this in the Petri net modeling methodology. The time can be directly associated with a place or it can be associated with the output transition of a place. For example, if a time of 10 time units is associated with activity p1 of Figure 10, it would indicate that that activity takes 10 time units. Alternatively, if a time of 10 time units is associated with transition a2, it would indicate that activity p1 must be marked for 10 minutes before it is completed. Associating time with places or with transitions can result in the same interpretation. In the "Timing Vector" input box of Figure 9, it can be seen that time is being associated with transitions. The user must enter this timing information by placing the number of time units in the "Timing Vector" input box.

We can now describe the dynamics of the simulation model. From the given initial state, a Petri net changes state by firing one or more of its transitions. A transition fires when it is enabled AND its timing vector has expired. A transition is said to be enabled

when all of its input places are marked. When the transition is enabled, the timing clock associated with that transition begins to run. When the clock expires, the transition fires. When the transition fires, it removes the tokens from its input places and puts tokens into its output places. Thus, the new state of the system is shown by the new marking of the places.

In order to run the simulation, the user enters the "Run Control" box. There are two parameters to enter. Under "Step Control", the user selects the number of changes of state that should occur before the simulation pauses. This is done by clicking the left mouse button on the appropriate circle. The current selection will appear as a black dot in the circle. In Figure 9, "Step 1" is selected. If "Step 5" is chosen, the simulation will run through 5 changes of state (transition firings) before it pauses. The user can enter any desired number of steps in the input box next to "Other". Under "Maximum Simulation", the user enters the maximum number of steps the simulation should run before it is finished. If this option is not selected, the simulation will run until it is ended by clicking the "End" button. At any time during the simulation, the simulation can be "Reset". Clicking the "Reset" button will reset all simulation counters to zero. After choosing the "Step Control" and "Maximum Simulation" options, the user begins the simulation by clicking "Start".

As the simulation runs, the simulation statistics appear on the simulation screen. The "Current Simulation Time" block at the lower left of Figure 9 is a running total of the amount of simulation time that has elapsed since the simulation began. The "Previous Marking / Current Marking" block at the upper right of Figure 9 shows each place (activity) that is marked (in process) at the last step and current step of the simulation. Since Figure 9 is the state of the simulation at current simulation time t=0, there is no "Previous Marking". The "Firing Vector" shows the number of times that transitions fire during the simulation run.

In Figures 9 and 10, at time t=0, the system is in its initial state as shown by the initial marking. At time t=1, the timing vector of transition a1 expires and transition a1 fires. The system moves to the state shown in Figure 11a and 11b.. In this state, activity p1 has begun and capacity c1 has been consumed into activity p1. Since p1 and c2 are the input places of p2 and both are marked at t=1, the clock for transition a2 can now begin timing. Transition a2 times out after 2 time units; i.e., at t=3. At that point transition a2 fires, removing tokens from p1 and c2 and placing tokens in p2 and c1. The new state of the system, Figures 12a and 12b, shows that capacity c1 is now available and that activity p2 is in progress. Transition a1 begins timing again and, after 1 time unit has elapsed, transition a1 fires again at t=4. This is shown in Figures 13a and 13b. At that time, activity p1 begins again. At this point both p1 and p2 are in progress. Finally, transition a2 and a3 are scheduled to fire at t =6. Since only one transition will fire at each step, it will take two steps to complete the firing of both transitions. The first firing is shown in Figures 14a and 14b and the second firing is shown in Figures 15a and 15b. Therefore, at t=6, the system has returned to the same state it was in at t=3. The circulation of tokens

and the marking of activities indicates the state of the system at various points in the simulation.

This brief introduction to Petri nets will give the user some insight as to how the simulation executes. The results of a simulation run can be interpreted in terms of the origional IDEF0 model. We will illustrate this in the tutorials of section 4. Users interested in more information on Petri nets and their use in modeling manufacturing systems are referred to references [1,3,5].

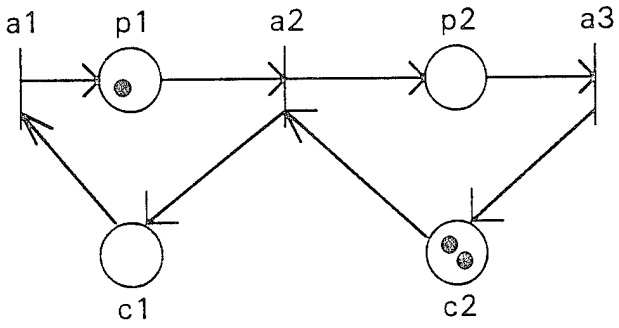


Figure 11a

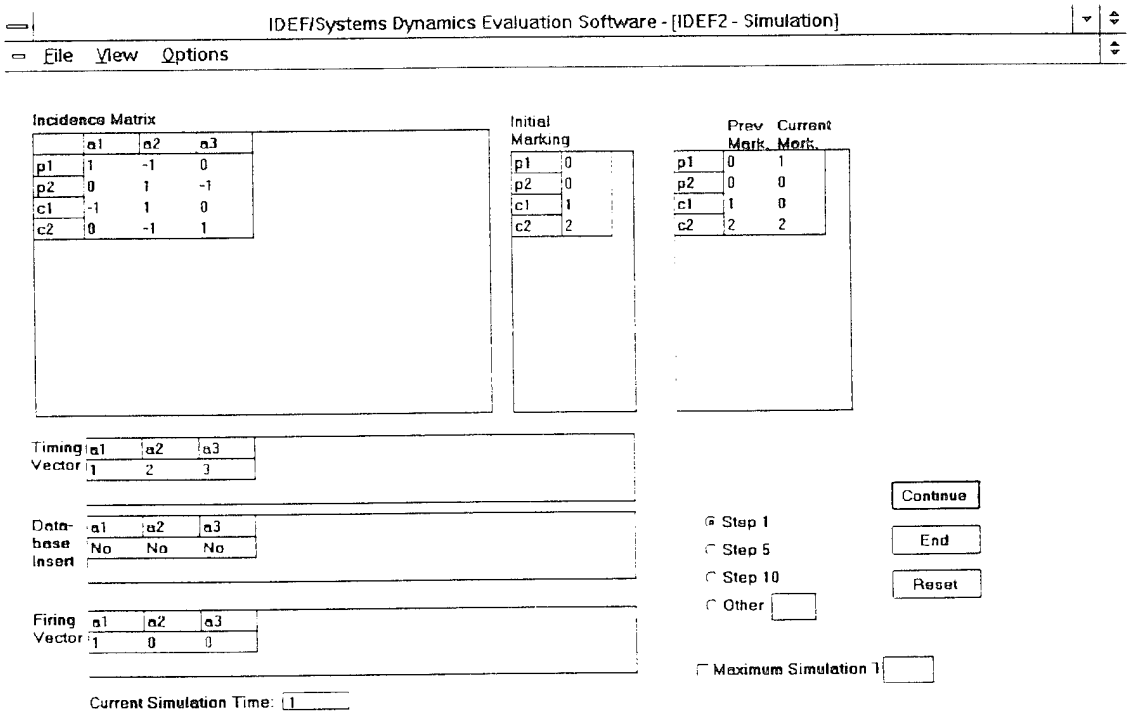


Figure 11b

Figure 11 State of the Simulation at Time t=1

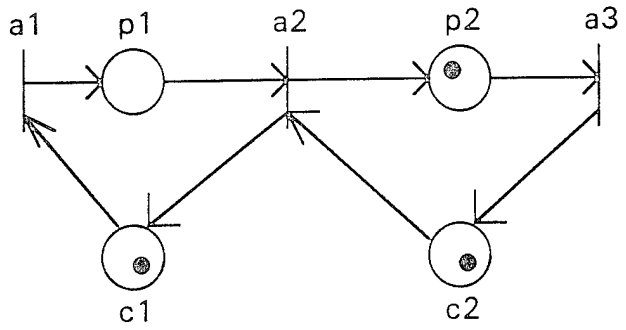


Figure 12a

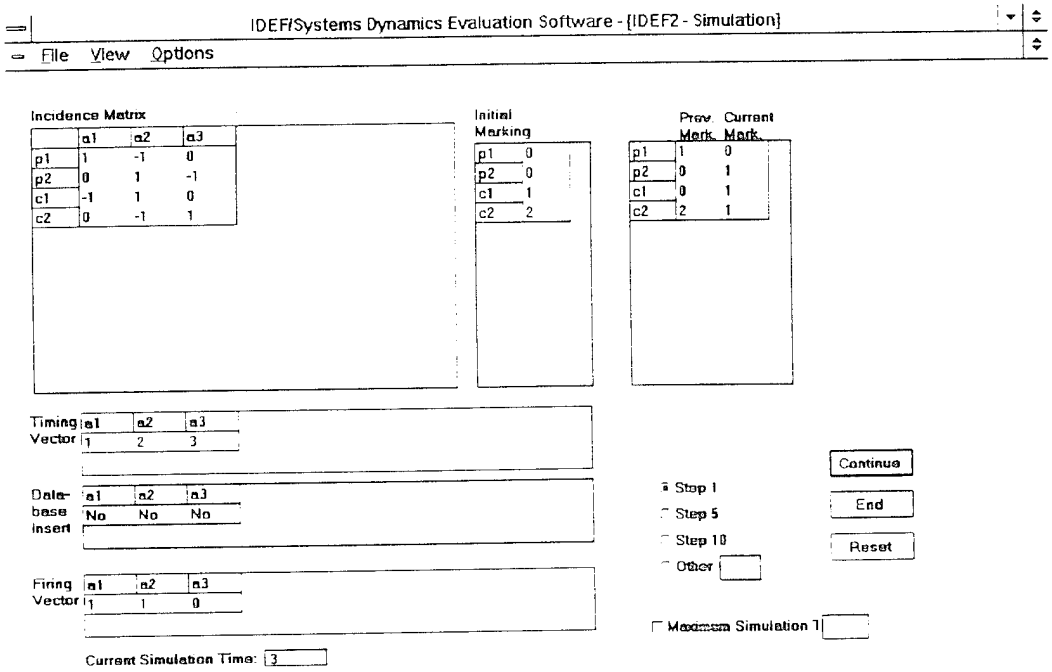


Figure 12b

Figure 12 State of the Simulation at t=3

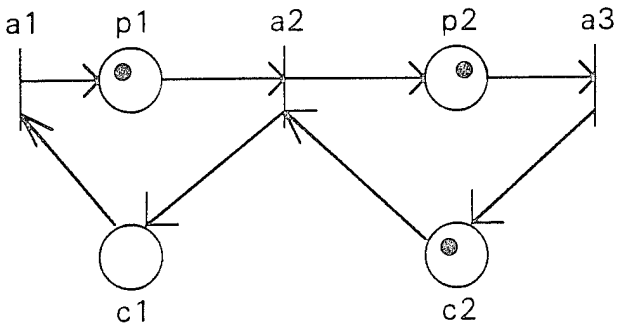


Figure 13a

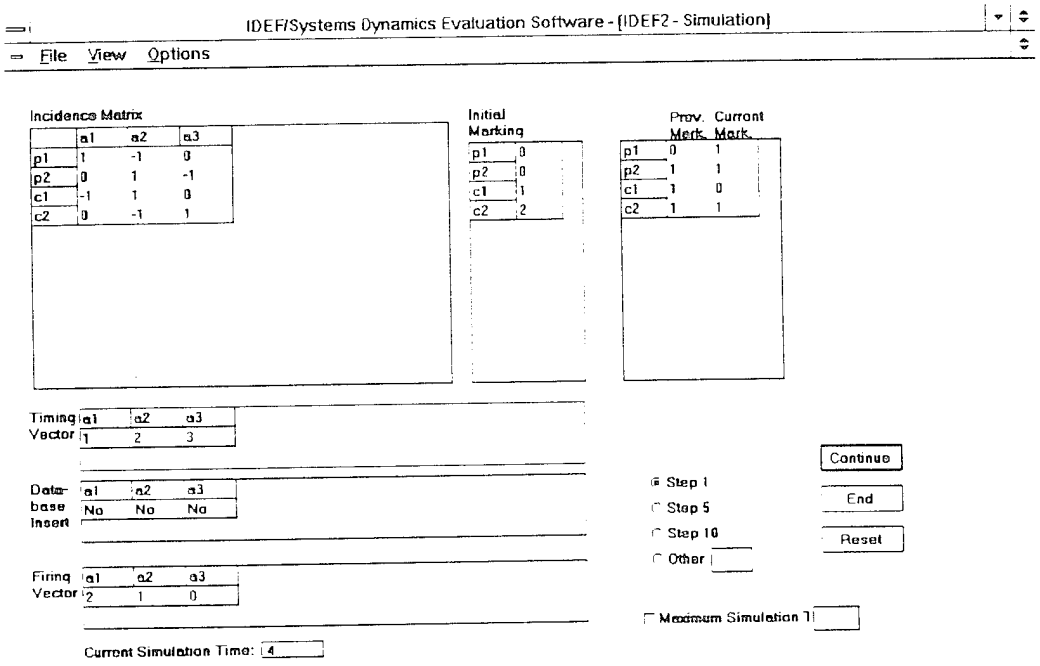


Figure 13b

Figure 13 State of the Simulation at t=4

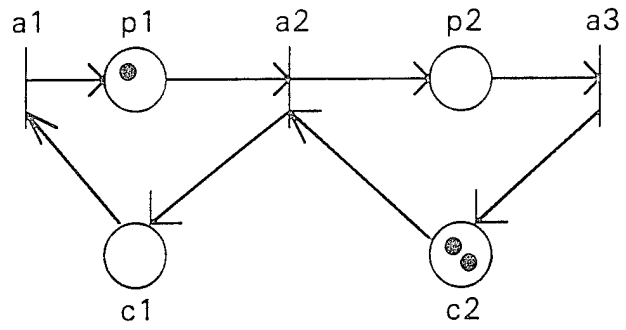


Figure 14a

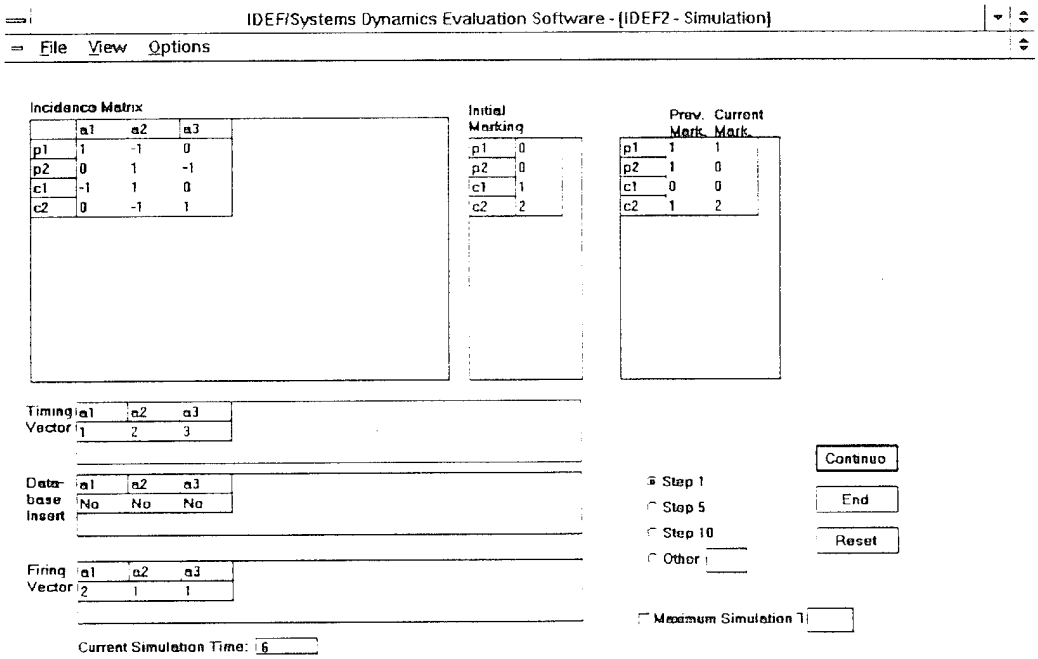


Figure 14b

Figure 14 State of the Simulation at t=6, First Firing

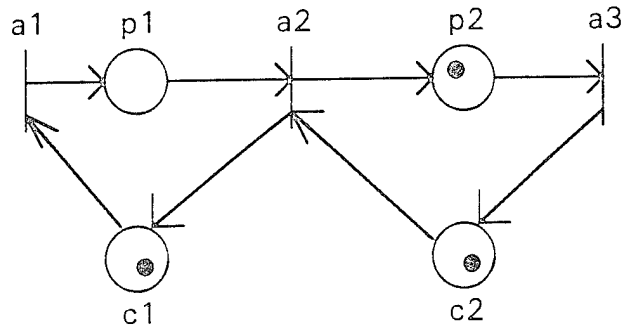


Figure 15a

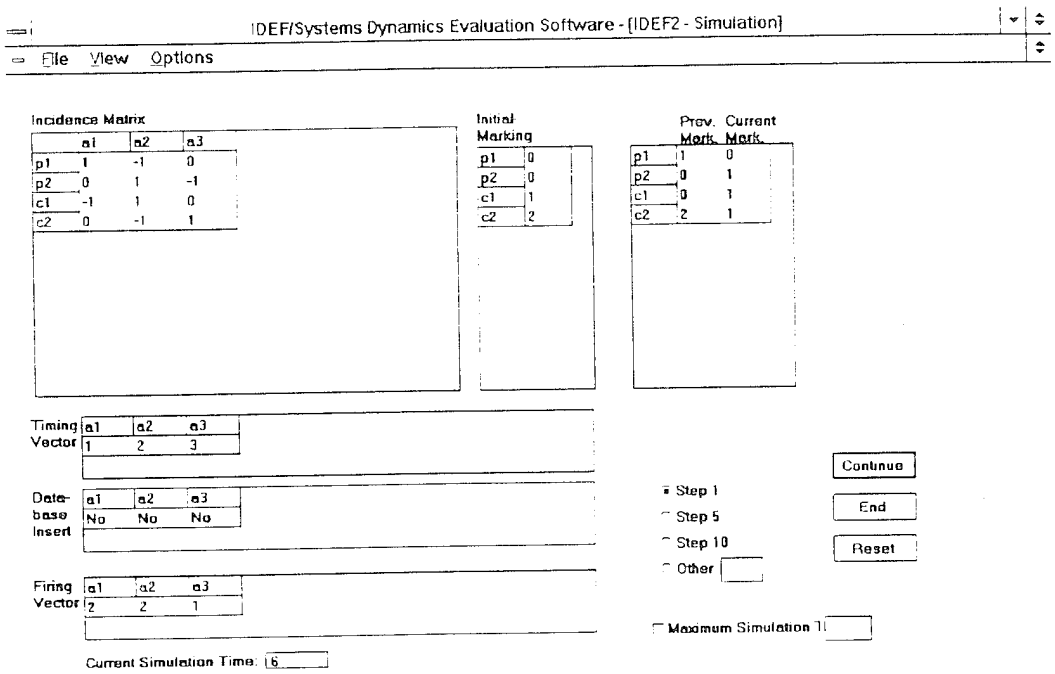


Figure 15b

Figure 15 State of the System at t=6, Second Firing

3.4 Summary

At this point we have described the two key screens for running IDEF0 / System Dynamics in software. These are the IDEF0 model input screen and the Simulation screen. In the next section we will describe a tutorial problem and lead the user through the running of a simulation.

4.0 TUTORIALS

There are two tutorials provided in the DATA directory. In this section we shall describe a tutorial program that simulates the operation of a packaging line and a tutorial program that simulates the operation of a robot cell.

4.1 Problem Description : Packaging Line Example

Figure 16 is the IDEF0 model of the operation of certain activities along a filling/packaging line. This model can be extended to include other operations and functions of the production activities. For illustrative purposes we limit our attention to the filling operation, weighing operation and movement of the package, called the "tray", along the line.

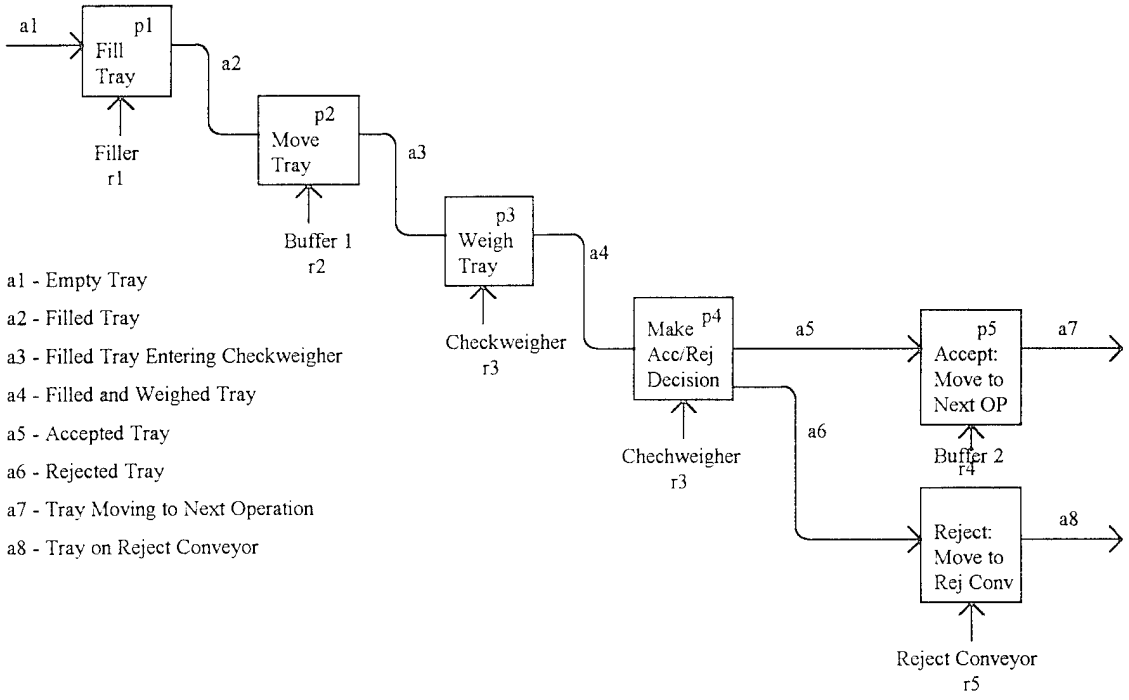


Figure 16 IDEF0 Model of a Filling and Weighing Operation

Events begin when an empty tray enters the filler. The first activity, "Fill Tray", is performed by the resource called the "Filler". The filled tray then proceeds down the conveyor in the activity "Move Tray". From a modeling perspective, the resource being utilized by the tray is the buffer capacity on the conveyor that exists between operations. Therefore, the resource is given the designation "Buffer 1". The tray then passes over the checkweigher and the activity "Weigh Tray" takes place. This is followed by a decision making activity "Make Accept/Reject Decision". It is important to note that one should separate each activity of the activity sequence in modeling a problem. The accept/reject

decision is a separate step from the act of taking the weight of the tray. Thus, we differentiate between activity p3 and activity p4, even though both are performed in sequence by the same mechanism.

In the next set of arcs, we come to our first example of what is known as a "conflict". A conflict exists when there is a choice of moving from an activity to only one of two or more activities that are *exclusive* of one another. In Figure 16, the activity following p4 will be *either* activity p5 *or* activity p6. A tray cannot be both accepted and rejected at the same time. The IDEF0 methodology is unclear about how to represent the conflict state. We have proposed in reference [2] that OR conditions be represented by having two arcs with separate labels exiting the activity from which the conflict takes place. This labeling convention is used in this software. Note the separate labels a5 and a6 in Figure 16.

It should also be noted that activities p3 and p4 both use the same resource, the checkweigher. The model must reflect the physical operation of the system. The checkweigher first weighs the tray and then reports a decision. The checkweigher will not weigh another tray until it has reported the decision for the last tray. In effect, the checkweigher is occupied for both activities. In terms of IDEF0 modeling, it is a *shared resource*. That is to say, it is shared between two activities of the model. We will see shortly that it is important to identify the shared resources in order to obtain a proper simulation model.

4.1.1 Entering IDEF0 Data: Packaging Line Example

The Packaging Line Example is in the IDEF directory under the project name "Convey.idf". However, we suggest that the user enter the problem data by following the instructions in this and the following paragraphs. This will give the user a better feeling for the process of entering a problem.

The first step is to start the software by double clicking on the IDEF icon and maximizing the screen. The project screen will appear as previously shown in Figure 1. Click on the *Project* menu and choose *New Project*. The "New Project" dialog box will appear as previously shown in Figure 2. Click on *Project* and *Save Project As*.. Type the project name "Filler.idf". The software will not automatically give the extension ".idf". You must type the extension. The "Untitled.idf" label will now change to the project name. Click the left mouse button in the panel labeled "Untitled.id0". A dialog box will appear requesting the name of the IDEF0 file. Type in the project name "Filler.id0". The software will not automatically give the extension ".id0". You must type the extension. Click on the command button "OK" and the new IDEF0 filename will be established. The "Untitled.id0" label will now change to your filename.

To enter the IDEF0 model data, click on the menu item "Edit". Three choices will appear: "IDEF0", "IDEF1X", and "Input/Output". Click on "IDEF0" and the IDEF0 data input screen will appear with the name of the project at the top.

Figure 17 shows the data as it will appear when the input steps are completed. The data is partially obscured. Using the scroll bars, data can be moved within the windows. The data requirements will be listed in the following paragraphs as we go through the steps of creating that screen.

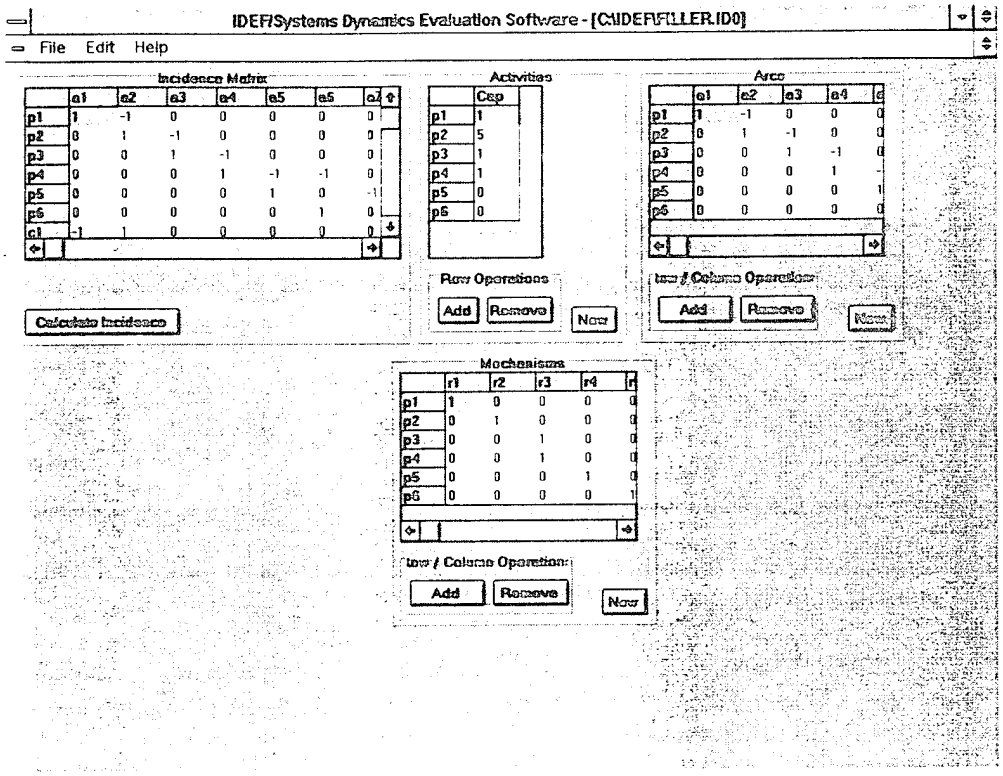


Figure 17 Input Data Screen for Filler.idf

Go to the "Activities" input box and click on the "New" command button. A dialog box will appear with the title "Rows" and the request "Specify the number of rows". The user is being asked for the number of activities in the IDEF0 model. The activities of the IDEF0 model will become the rows of the incidence matrix. At this point type "6" because there are six activities in the IDEF0 model of Figure 16. Click on "OK" and six activities, labeled p1 to p6, will appear in the activities box.

Now enter the capacities for each of the activities. Referring to Figure 16, the first activity, p1, can only be performed on one tray at a time. Hence, it has a capacity of "1". Click on the capacity cell of activity p1 with the left mouse button. This will access the

cell. Click once with the right mouse button and the contents of the cell will be incremented to "1". The second activity, p2, is the tray moving in the buffer. For the purposes of this exercise we will assume a capacity of "5". That is, there will be buffer capacity for five trays between the filling station and the weighing station. Click on the cell with the left mouse button to enable it and click five times on the cell with the right mouse button to add a capacity of "5". Similarly, go to the checkweigher cell (p3) and enter a capacity of "1". This also applies to activity p4.

If the capacity of an operation is not limited, this is indicated by leaving the cell blank. We will assume that there is no downstream blockage on the take away conveyors for the accept and reject lanes. Therefore, their capacity cells will remain blank. We could enlarge the model by adding all the downstream activities. In that case, we would have to indicate the capacities of conveyor buffers between those activities that are further downstream. At this point the activities data has been entered.

Go to the "Arcs" input box and click on the "New" command button. A dialog box will appear with the label "Columns" and the statement "Specify the number of columns". The user is being asked for the number of arcs, which will become the columns of the incidence matrix. Referring to Figure 16, there are 8 arcs, a1 through a8. Therefore, enter the number "8". The arc labels will appear across the columns of the arcs matrix and the activities will appear down the rows. The user must now enter the relationship between activities and arcs in the "Arcs" input box matrix.

Figure 18 shows the complete matrix of inputs required. The relationship between activities and arcs is given by the IDEF0 model of Figure 16. If an arc is an input arc to an activity, a "1" is entered in the cell intersecting the row with the activity label and the column with the arc label. If an arc is an output arc of an activity, a "-1" is entered in the cell intersecting the row with the activity label and the column with the arc label.

The reader should enter the relationships as shown in Figure 18 and check the data against the model of Figure 16 in order to confirm the relationship. Click on the left mouse button to access a cell. Then click on the right mouse button to increment the value in the cell by one and click on the left mouse button to decrement the value in the cell by one. When all the data of Figure 18 is entered, the arc data entry is complete.

	a1	a2	a3	a4	a5	a6	a7	a8
p1	1	-1						
p2		1	-1					
p3			1	-1				
p4				1	-1	-1		
p5					1		-1	
p6						1		-1

Figure 18 Data Entry for "Arcs" Input Box

Go to the "Mechanisms" input box at the lower right of the screen. Click on the "New" command button. A dialog box appears that asks for the number of mechanisms. There are five mechanisms (or resources) being employed in Figure 16, r1 through r5. Input the number "5". Click "OK". The mechanisms input box will list the mechanisms as column labels. Go to each cell of the mechanisms input box and establish the relationship that exists between the activities and their mechanisms. If a resource is a mechanism of an activity, place a "1" in the cell that is the intersection of the activity label and the mechanism label. Otherwise, leave the cell empty. This is done by first accessing the cell with the left mouse button and incrementing the cell with the right mouse button. The complete matrix of entries is shown in Figure 17. Note that there are two entries under resource r3, the checkweigher. This resource supports two activities: p3 and p4. This is a shared resource and will have a special significance in the simulation. This will be discussed in a late section.

Go to the "Incidence Matrix" input box in the upper left of the screen. Click on the "Calculate Incidence" command button. The incidence matrix is automatically calculated from the data that has been supplied. Note the arc labels along the top and activity labels down the rows. In addition, capacity places are added as c1 to c4. These are the capacities of the first four activities. Also, the shared resource, r3, is included as an activity.

At this point all the data has been entered for the simulation. Click on the "Files" menu and click on "Save As". You will be asked again to provide a filename. Type "Filler.id0" and click "OK". The IDEF0 model is now saved under the filename "Filler.id0". Exit the IDEF0 input screen by clicking on *File* and *Exit*. The next step is to simulate the model.

4.1.2 Simulation: Packaging Line Example

Enter the simulation screen by clicking on the menu item "Analysis" and choosing "IDEF2-Simulation". A screen shown with the layout of Figure 19 will appear, with the incidence matrix provided at the upper left of the screen. In order to clarify the meaning of the incidence matrix for this problem, we have diagrammed its Petri net structure in Figure 20.

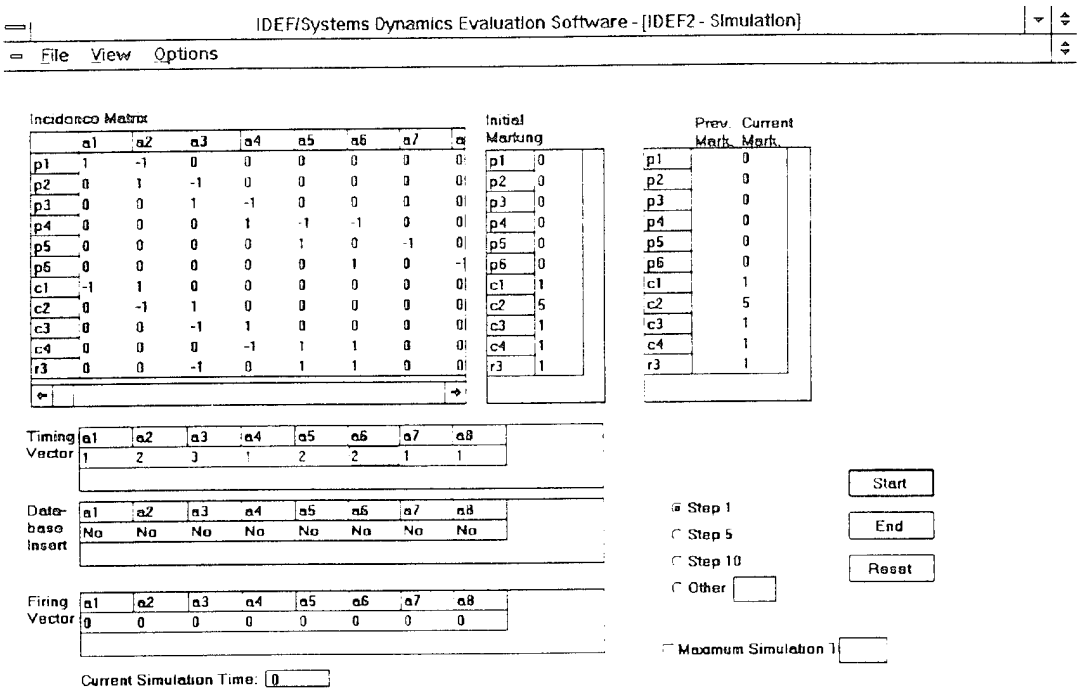
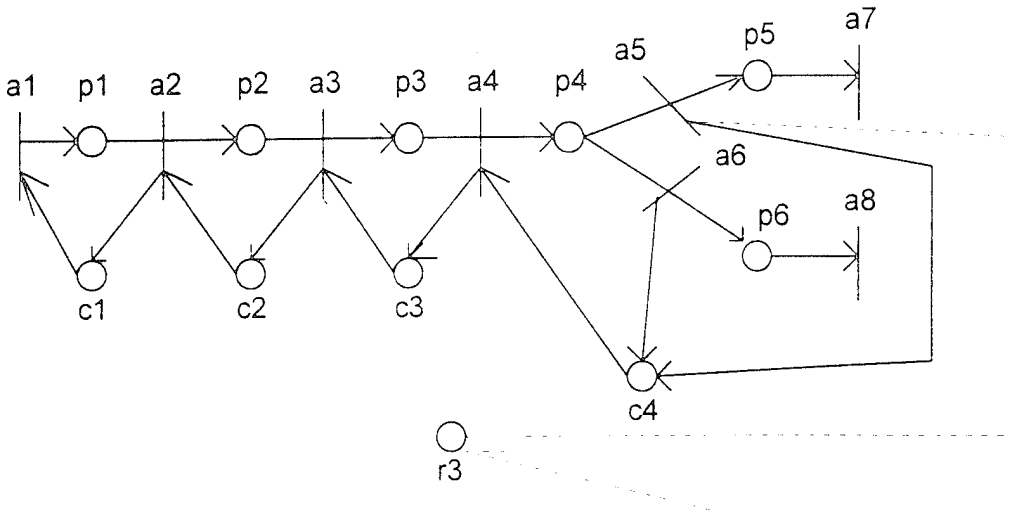


Figure 19 IDEF2-Simulation Screen for Filler.id0

Figure 20 maps the flow of the IDEF0 model of Figure 16. In addition, it includes the capacity holders as complementary places of the activities for which capacities have been specified. The shared resource, r3 (checkweigher) is shown entering the main flow at a3, which enables p3 when fired. The activity p3 is the first activity for which the checkweigher is required. The resource r3 is shown exiting the main flow at a5 and a6. These are the events which occur after p4, which is the last activity for which the checkweigher is required. A token placed in r3 will circulate through p3 and p4 and back to r3 as the simulation runs. Since both p3 and p4 require that token from r3, either p3 OR p4, but not both, can be active at one time. In effect, the shared resource prevents more than one tray from being in the checkweigher at a time. Note that, if r3 were not present, c3 and c4 could not guarantee that a token would not be in p3 and p4 simultaneously. This is the special role of the shared resource place. In fact, c3 and c4 are not needed for this simulation. They were included simply to make this point about the special significance of shared resources in the IDEF2 simulation.



a1 - Empty Tray
a2 - Filled Tray
a3 - Filled Tray Entering Checkweigher
a4 - Filled and Weighed Tray
a5 - Accepted Tray
a6 - Rejected Tray
a7 - Tray Moving to Next Operation
a8 - Tray on Reject Conveyor

p1 - Fill Tray
p2 - Move Tray
p3 - Weigh Tray
p4 - Make Accept/Reject Decision
p5 - Accept: Move to Next Operation
p6 - Reject: Move to Reject Conveyor
c1 - Filler Capacity
c2 - Buffer 1 Capacity
c3, c4 - Checkweigher Capacity
r4 - Checkweigher resource

Figure 20 Petri net for Filling and Weighing Operation

At this point the user should initialize the simulation model. Figure 19 shows a complete set of data entered. Note the the initial marking is the capacity of the capacity places and the shared resource. The "Timing Vector" was selected arbitrarily. Enter the data as shown in Figure 19. This should be followed by running the simulation in "Step 1" increments. Figures 21 through 27 show the simulation at various stages. The stages are given by the number of steps the program has executed. The user should step through the simulation and compare the screens with the figures provided. The descriptions beneath the figures give the number of steps and a brief description of the events. Events can be followed by comparing the "Current Marking" with the "Previous Marking" at each step.

The user should note the transition from step 7 to step 9. Here, the checkweigher has executed the conflict condition by accepting the tray. In its current form, the software executes conflicts using what is called a *race* condition. In effect, when there is a choice, the event that will occur first depends on which transition fires first. The transition that fires first is the one whose time expires first. Transitions a5 and a6 both have the same time. When this is true, the software evaluates the lower number first. A future enhancement of the conflict condition would be to allow the user to set probabilities for the firing of the conflicting transitions. This is not available in this version of the software.

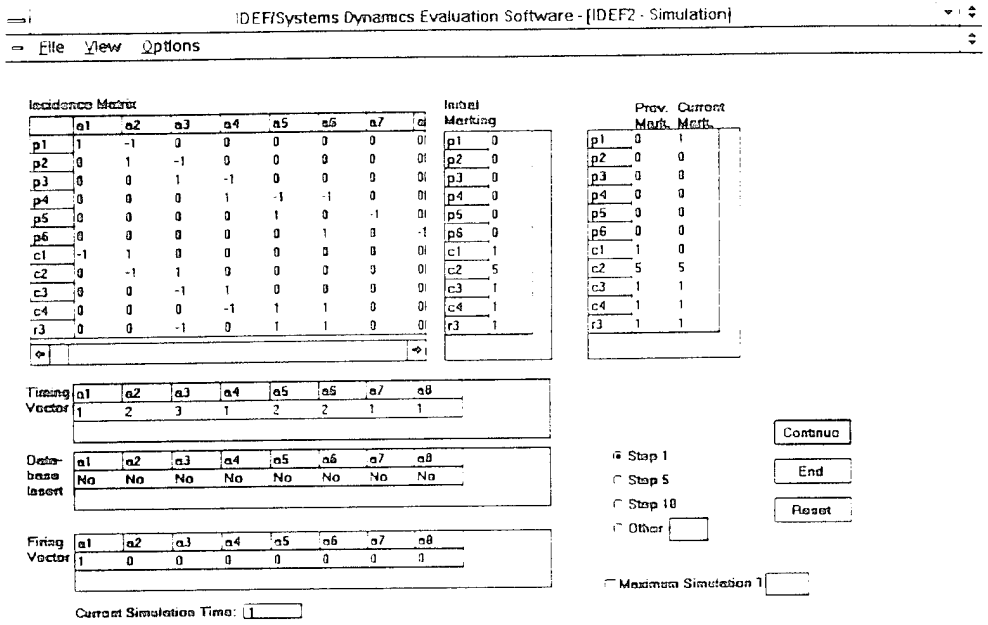


Figure 21 Step 1: Tray Enters the Filler

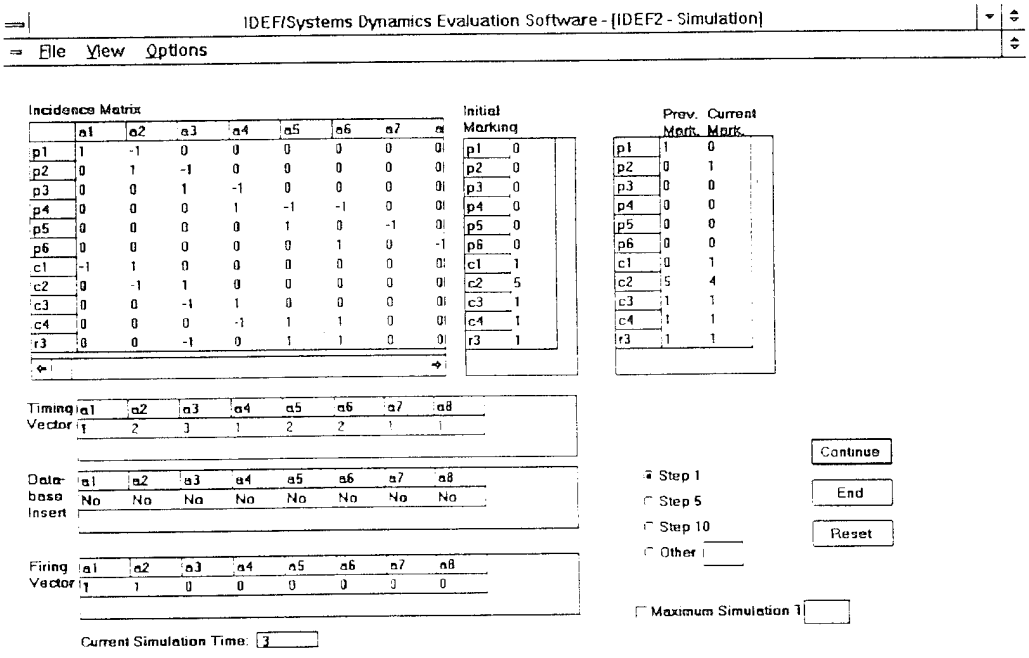


Figure 22 Step2: Tray Leaves Filler, Enters Buffer 1

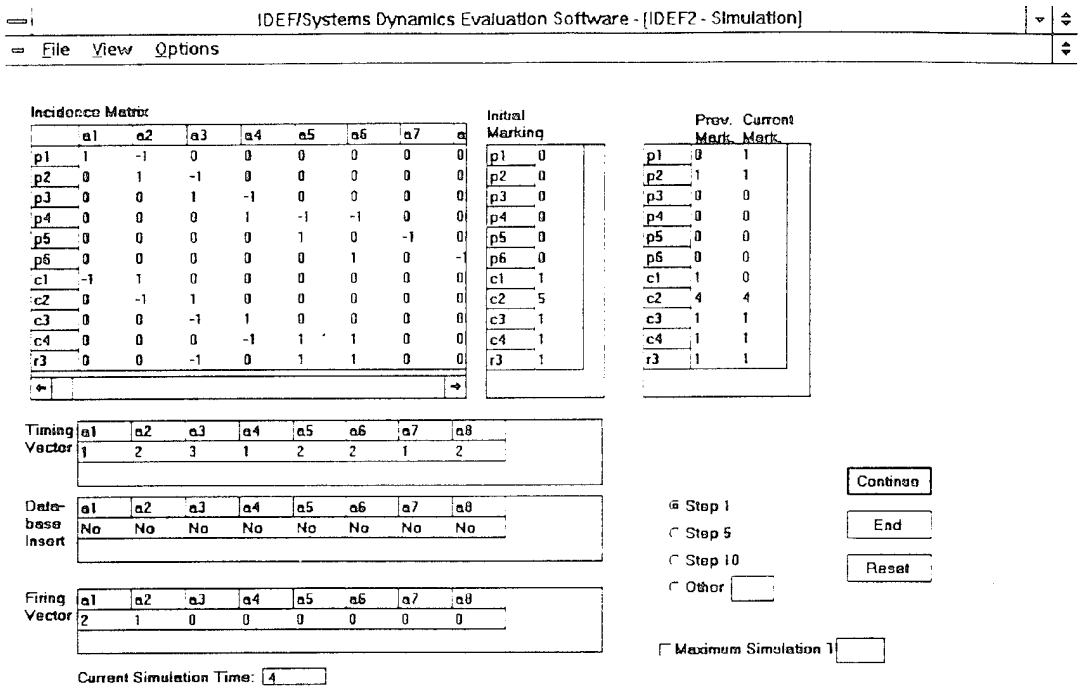


Figure 23 Step 3: New Tray Enters Filler

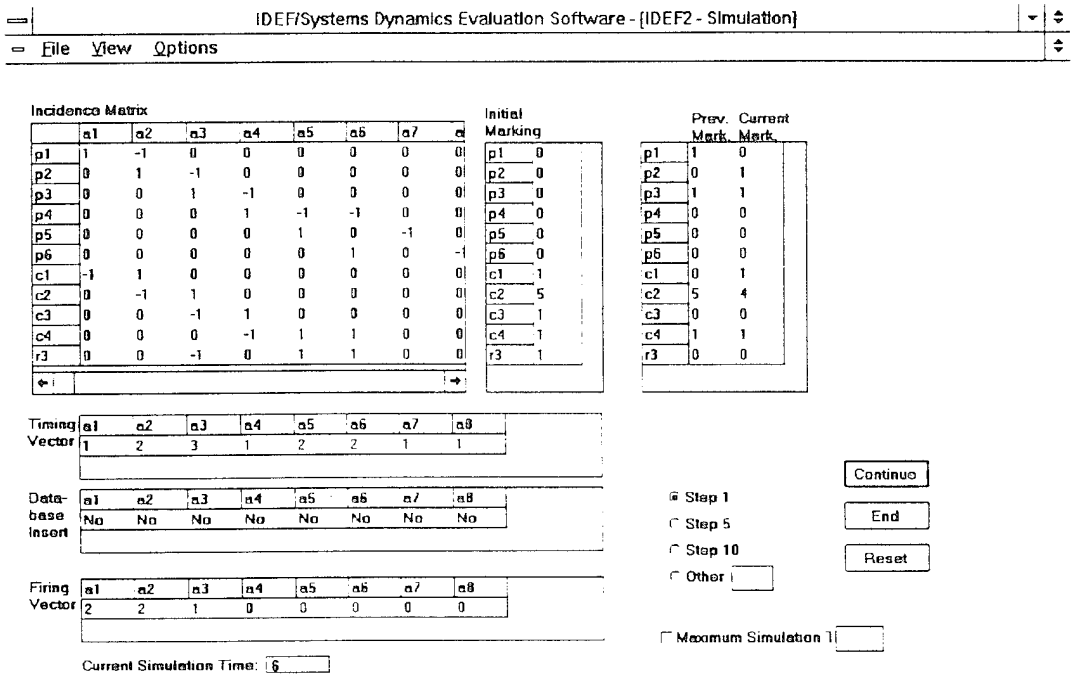


Figure 24 Step5: Tray Enters Checkweigher and Tray Enters Buffer 1

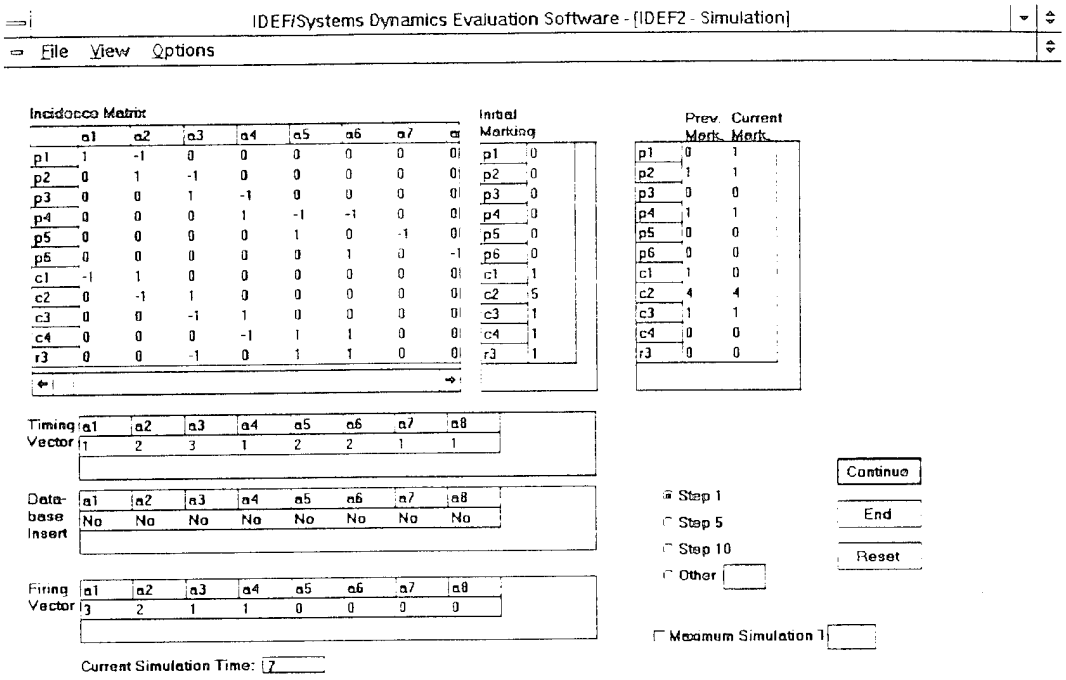


Figure 25 Step 7: Tray Enters Filler and Tray in Checkweigher Accept/Reject

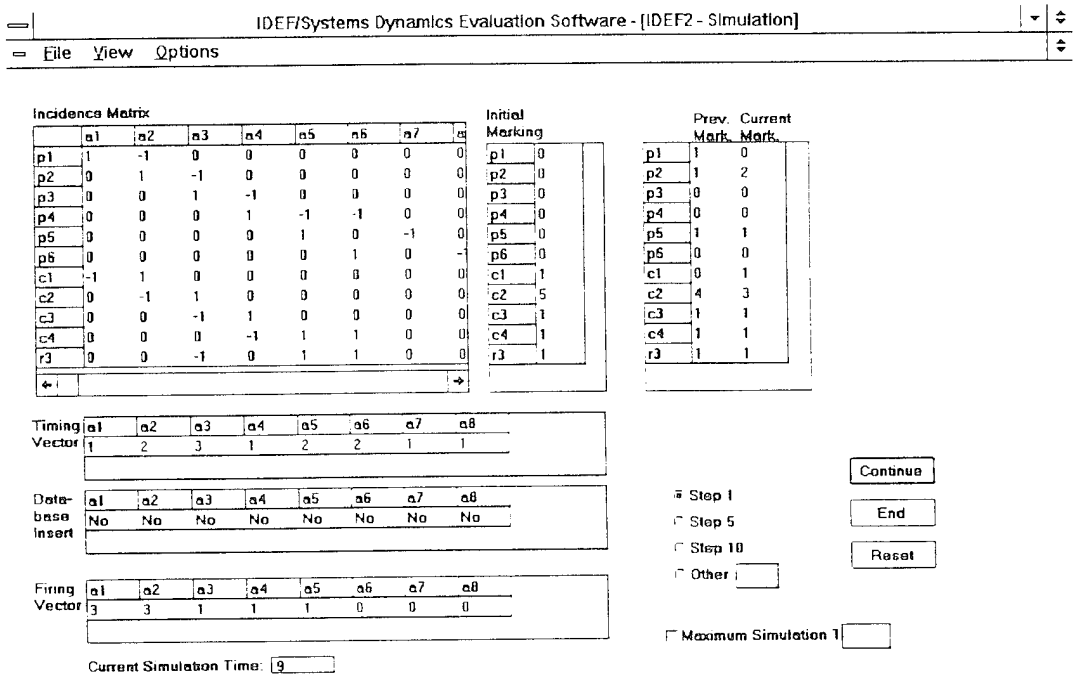


Figure 26 Step 9: Two Trays in Buffer 1 and Accepted Tray Moving to Next Operation

4.2 Problem Description: Robot Tending Lathe

In this example we will illustrate some additional points about the construction and interpretation of IDEF0 models in this software as well as illustrate the use of the simulation to interact with a database file. In this example we consider the case of a machining cell controller that is responsible for coordinating the activities of a robot and a lathe. Figure 27 represents the relationship between the cell controller and the controllers for the two machines. The activities of the IDEF0 model are the sequencing activities coordinated by the cell controller.

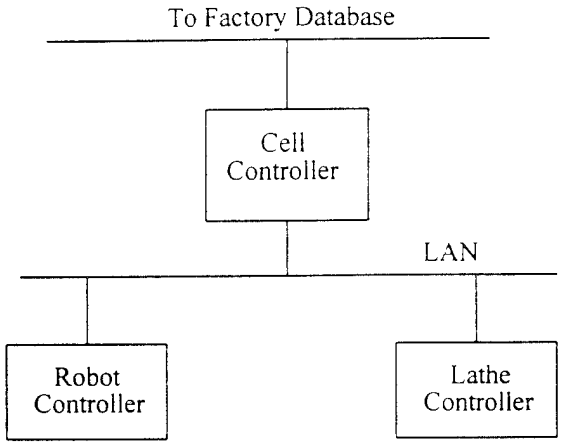


Figure 27 Machining Cell Communication Hierarchy

Figure 28 is the IDEF0 model of a robot tending a lathe in a machining operation. Events begin when a workpiece is put into the input buffer of the machining cell (activity p1). At that point the workpiece is available to be loaded onto the lathe by the robot (activity p2). Once loaded, the workpiece is turned by the lathe (activity p3). This is followed by the robot unloading the lathe (activity p4). Finally, a finished component is put into the finished part buffer (activity p5) and the cell controller logs to the database the fact that a part has been completed (activity p6).

As in the previous example, there are resources that are shared among activities; i.e., the lathe and the robot. Unlike the previous example, there is no conflict. When activity p4 is completed, both of the succeeding activities (p5 AND p6) are activated. This is an example of *parallelism*. Parallelism occurs when an arc exiting one activity divides and enters two or more succeeding activities. In order to indicate parallelism within this software, the dividing arc has only one label, in this case a5. The reader will recall that, in the case of conflict, two arcs exit the first activity and are given different labels.

Another extension of the software illustrated in this example is the use of *activity objects*. An activity object is more than just another activity box. It has attached to it a

procedure which defines the purpose of the object. In Figure 28, the activity p6 ("Insert") is an activity object. When the activity is enabled, data is inserted into a specific destination in a data base. The programmer must designate the destination data table. The use of this activity object will be illustrated later.

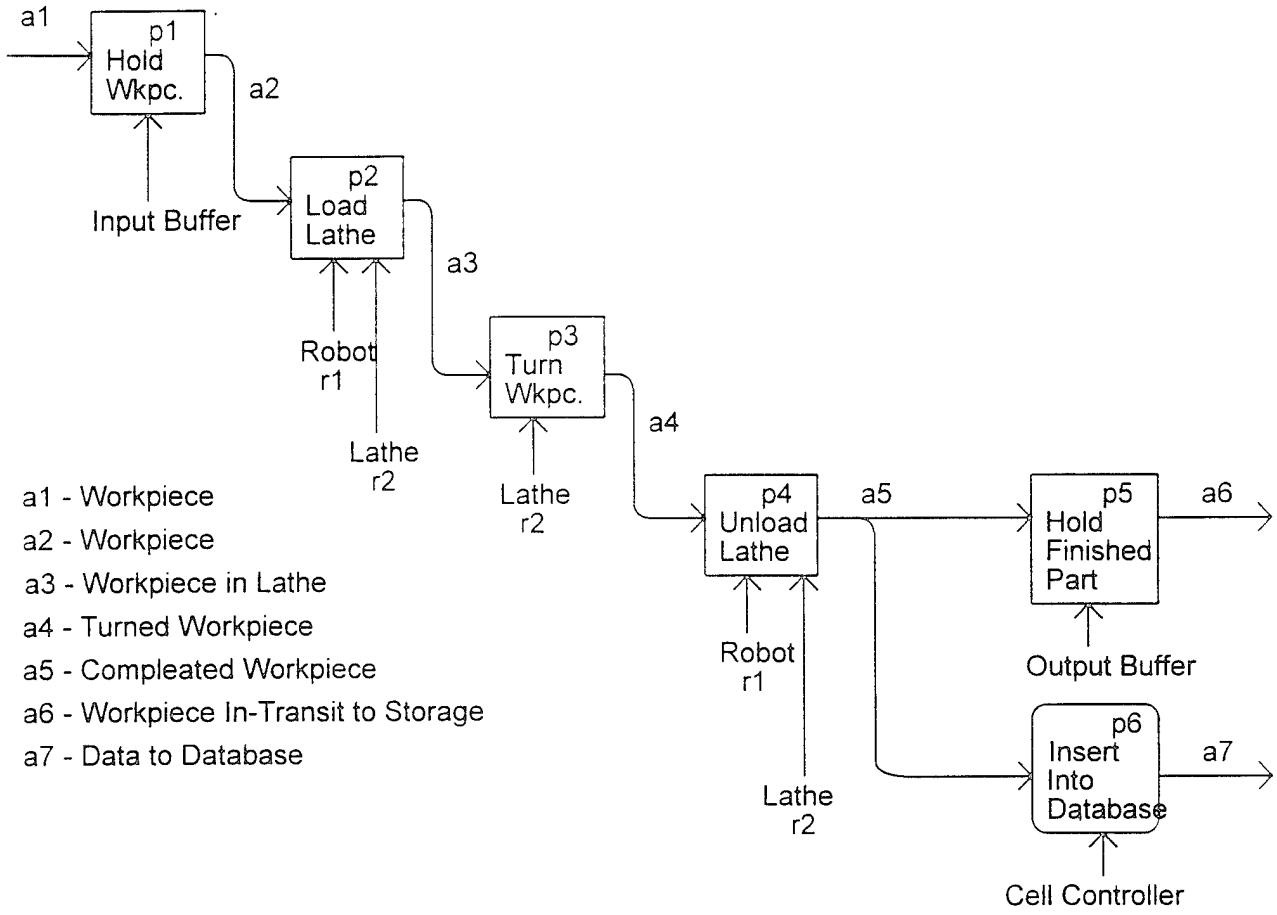


Figure 28 IDEF0 Model: Robot Tending Lathe

4.2.1 Entering IDEF0 Data: Robot Tending Lathe Example

The input data screen for the model of Figure 28 is shown in Figure 29. This example is in the IDEF directory and has the project name "Lathe1.idf". In reviewing this example, the reader will load it from the "Project" menu and "Open Project" submenu. The reader should start the IDEF software, click on "Project", select "Open Project" and select "Lathe1.idf". Click "OK" and the project name will appear on the screen. Click on "Edit" and "IDEF0". Figure 29 will appear on the screen.

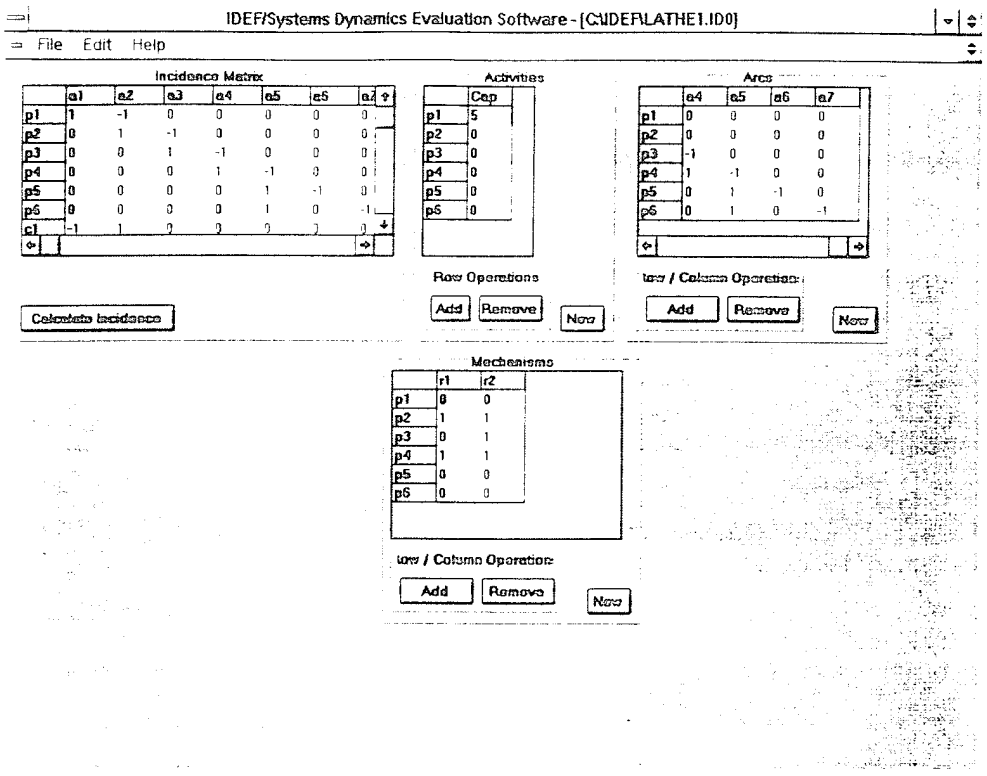


Figure 29 IDEF0 Input Screen: Robot Tending Lathe

From the previous example, the reader is familiar with the interpretation of the data in the input boxes. We will assume a capacity of 5 units in the input buffer (activity p1). Scroll through the other input boxes and confirm that the data entered reflects the IDEF0 model of Figure 28. The buffer capacities on the activities were chosen arbitrarily.

Note the data structure for parallelism in the "Arcs" input box (a5). This indicates an exit at p4 and an entrance at both p5 AND p6. Note the fact that there is no distinction made at this point between an ordinary activity (p1 through p5) and an activity object (p6). This distinction is made during the initialization of the simulation model, which is the subject of the next section.

4.2.2 Simulation: Robot Tending Lathe Example

The reader should exit the input screen and select *IDEF2-Simulation* under the *Analysis* menu. Figure 30 shows the simulation screen with the initial markings and timing vector. Here we assume that the machining operation takes 4 time units and all other operations take 1 time unit. The user should enter this information.

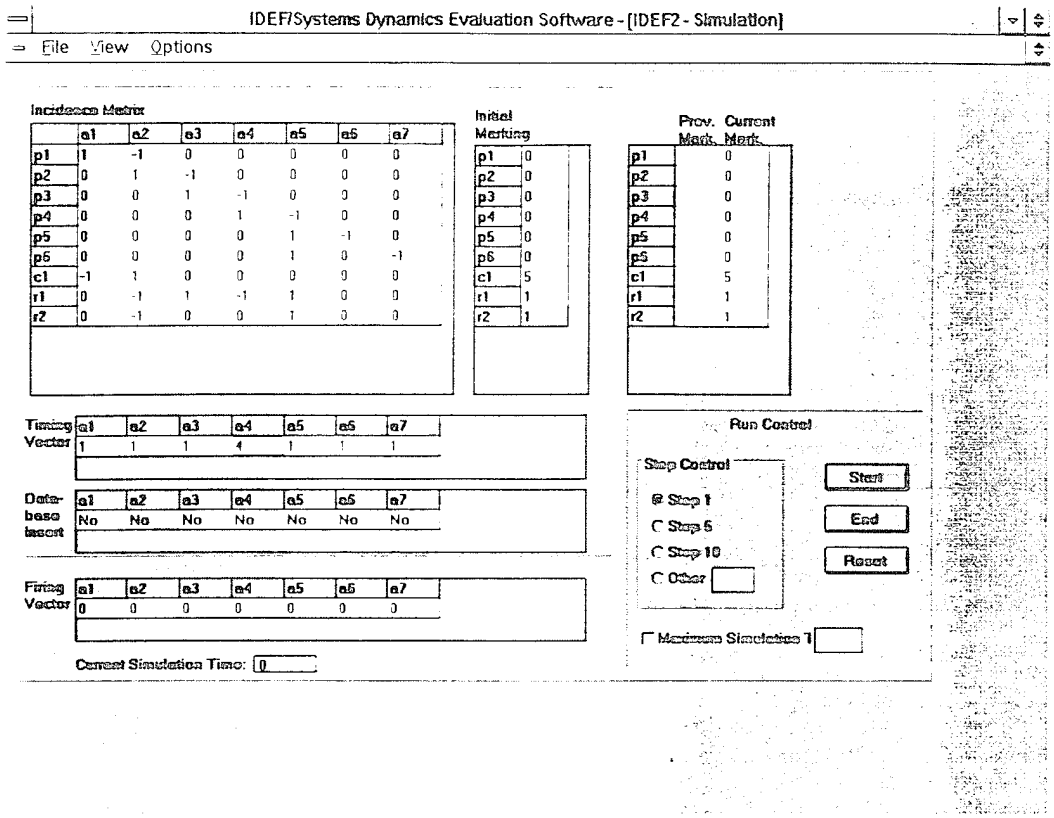


Figure 30 Simulation Screen: Robot Tending Lathe

In order to illustrate interaction with a database, it is necessary to establish connectivity between the model and a database file. A database file in Access format has been provided in the DATA subdirectory under the title New1.id1. The extension refers to IDEF1X. Referring to the IDEF0 model of Figure 28, the database insert occurs when p6 is complete. Alternatively, this is when a7 fires. Therefore, the database insert will be associated with transition a7.

Double click on a7 in the panel labeled "Database Insert". The dialog box shown in Figure 31 will appear. Click on "new1.id1" and "Table1" will appear. Click on "Table1" and "OK" and the software will return you to the screen shown in Figure 32. Note that you are now connected to the database file by the appearence of "Yes" in the a7 cell of the Database Insert panel. The title "new1.id1" is the name given to an Access database file. "Table1" is the name of a table in that file. That table has been previously established and contains three attributes, or record fields. The insert object has been written with a procedure that simply loads three random numbers into the database table records each time the insert object executes.

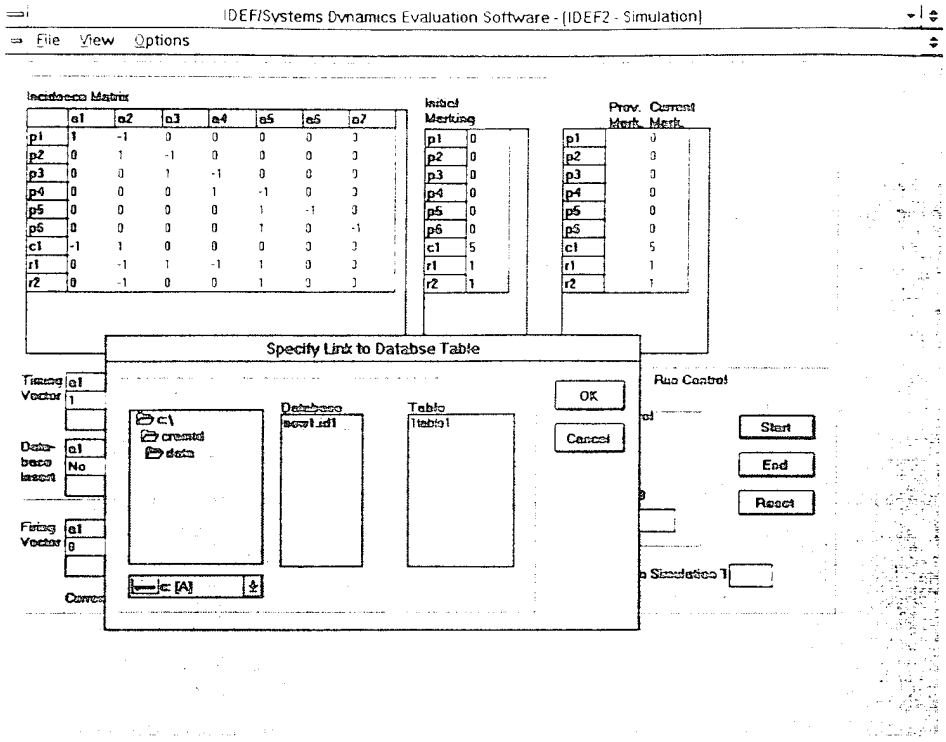


Figure 31 Specify Link to Database Table

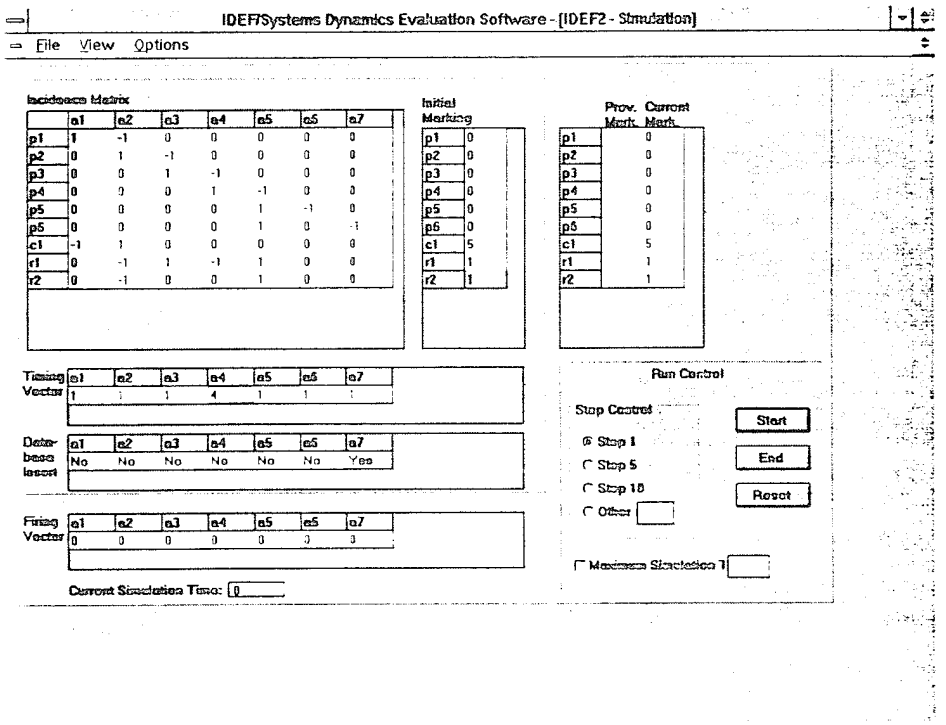


Figure 32 Link Made with Database Table

Start the simulation by clicking on the *Start* button. Continue the simulation until the firing vector cell a7 fires once. At that point an entry has been made into "new1.id1". In order to see the entry, click on the menu item *View* and the submenu item *Database Insert*. A screen like that of Figure 33 will appear. Click on *1Table1* and the current contents of the database file will appear. Return to the simulation screen and continue the simulation until a7 fires two more times. Return to *View* and verify that two more entries have occurred.

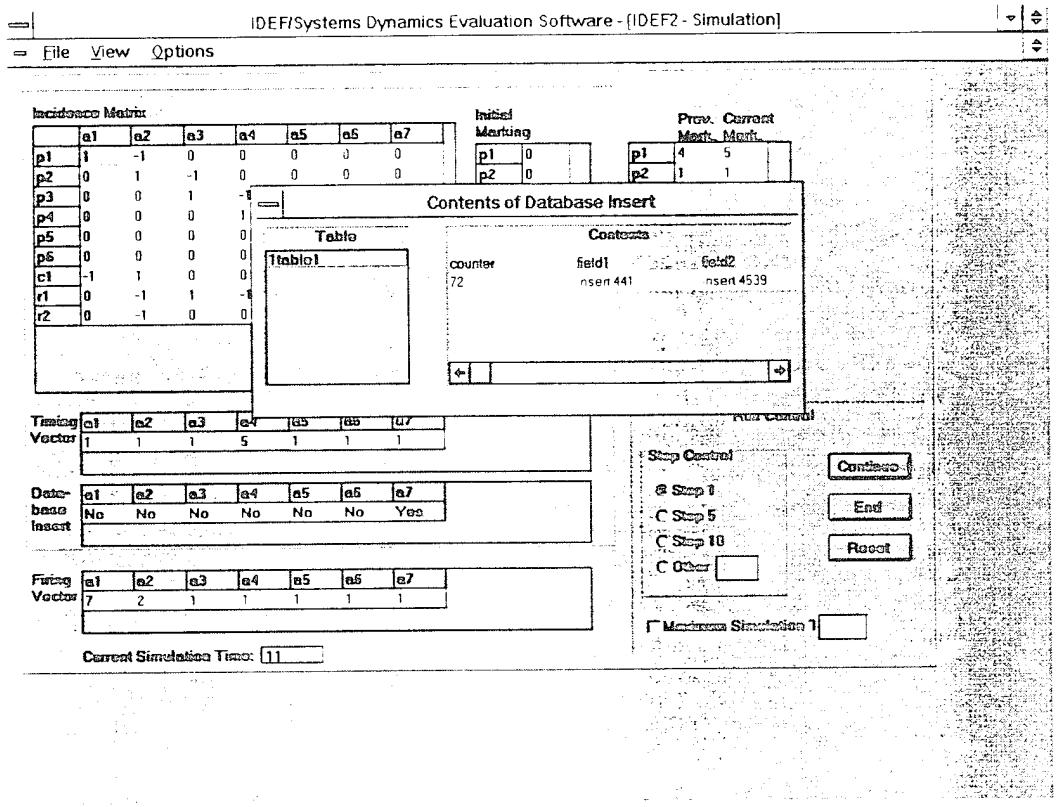


Figure 33 Contents of Database Insert

The user may create his/her own database tables from the project menu. Return to the Project menu and click on *Edit*. Select *IDEF1X* submenu item. A screen such as that shown in Figure 34 will appear. Here the filename "Part.id1" has been entered. Any filename can be entered as long as the extension ".id1" is included. Enter a filename with the extension "*.idf". Click "OK" and the user will enter the screen shown in Figure 35. Click on *Draw*, then *Entities*, then *Independent*. This will put the user into a screen such as that of Figure 36. The user should enter the name of the entity. This will become the name of the database table. Here the name "Parts" is entered. Clicking on "OK" puts the user into the screen shown in Figure 37. The user must now enter the name of the attributes, or records that will appear in the table. One of these attributes should be the

primary key. By entering the name in the top panel and selecting a key type, if appropriate, and clicking on *ADD*, the attribute name is entered. In Figure 37 we have entered two fields. Click on EXIT and Figure 38 will appear. This shows the table structure with the entity name at the top and the attribute names within the table. It is left as an exercise for the user to return to the simulation model, connect a Database Insert cell to the file "Part", execute the model and observe that the insert occurs.

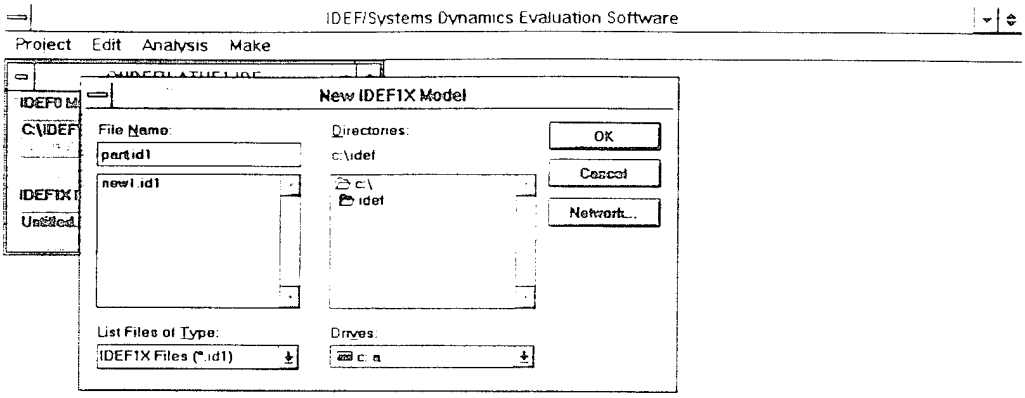


Figure 34 Establishing a New IDEF1X Model

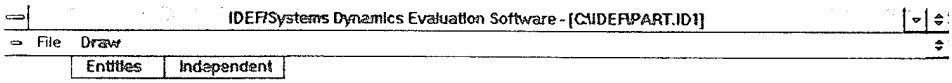


Figure 35 IDEF1X Data Entry Screen

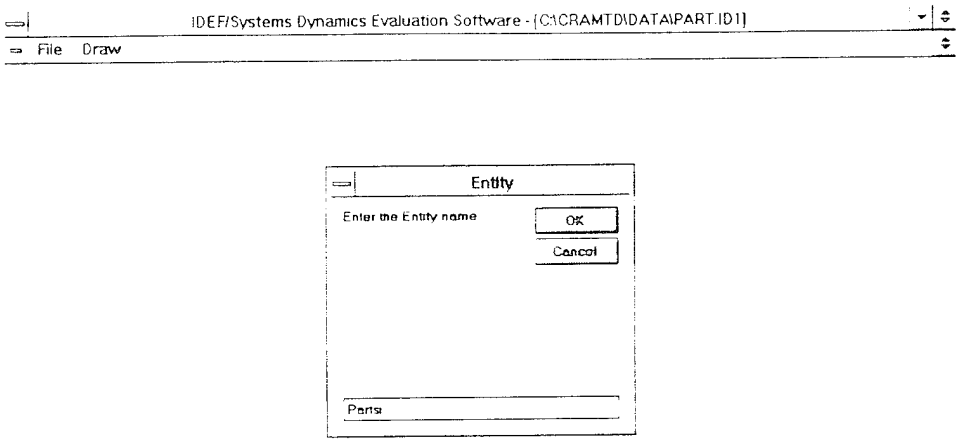


Figure 36 Entering IDEF1X Entities

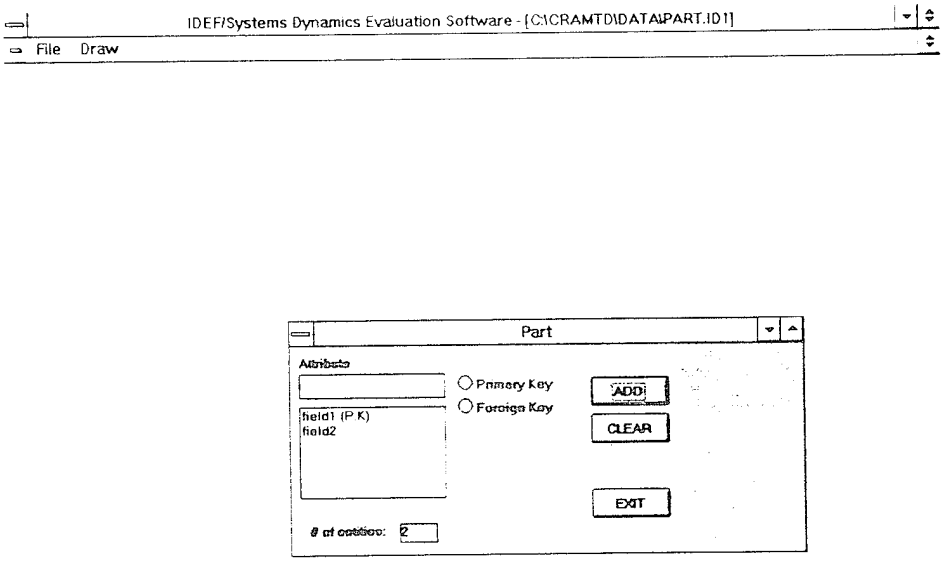


Figure 37 Entering IDEF1X Attributes

The database files are in the Access database format. In order to clear the records entered, the user can go into Access and open the file. Access uses the extension ".MDB" on database files, so it is necessary to select the *list all files* option and select the appropriate ".id1" file to open.

It is important to point out that the work done in establishing database connectivity is to test feasibility only. Thus the use of random numbers to show the connectivity was used. In further development there are several uses to which this can be put. One use is to record each event of the simulation and the simulation time at which it has occurred. By so doing, the database would contain the entire record of a simulation run, providing the data to be used in summary statistics, such as cycle times, queue lengths and output rates. A second use is to run the simulation to a particular schedule by providing the sequence of products to be produced in a data table, having the simulation enter product types into the simulated manufacturing system from the table and reporting completions to the database. These applications will require further development. However, the basic methods and concepts, as well as the connectivity with IDEF0, has been established in this prototype software.

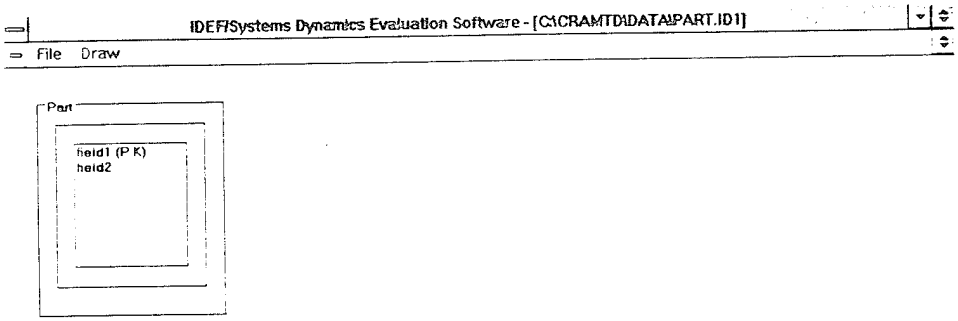


Figure 38 Database Table Created by User

5.0 OTHER FEATURES OF IDEF/SYSTEM DYNAMICS SOFTWARE

There are three additional features of the software. These are the *Policy Option* feature of the simulation model, the ability to compute *Invariants* of the simulation model and the ability to develop *ladder logic* representations of the boolean control logic in cases where the problem being modeled is an automatic control problem. In the following sections we will describe each of these features.

5.1 Simulation Policy Options

Refer to Figure 39. The menu item labeled *Options* allows the user to select a policy for the firing of transitions. Click on the *Options* menu and click on the *Policy* submenu. A dialog box will appear such as that shown in Figure 39. There are two policies to choose from: 1) Priority on event time and 2) Priority on enable. These can explained with reference to Figure 40.

There are cases where a conflict can occur in a simulation and that conflict must be resolved by a firing policy. *Priority on event time* settles conflicts by allowing the transition that times out first to fire first. Assume in Figure 40 that p1 obtains a token at t=0. Since the transition a1 requires only a token in p1, the transition a1 is scheduled to fire in 4 time units. Let us suppose that, at t=2, a token arrives in p2. Since a2 requires a token in p1 and p2 to fire, it is now scheduled to fire in 1 time unit; i.e., at t=3. Since both transitions require the token from p1, only one of the transitions will be allowed to fire. The user selected policy determines which token will fire. *Priority on event time* allows a2 to fire first because it will time out first. *Priority on enable* states that the transition that is first enabled captures the token at that time and that transition will fire after its timing vector times out, regardless of whether or not another transition is enabled later and requires the same token. This policy would have a1 fire first because it was enabled first by the arrival of a token in p1.

The appropriate policy is a function of the interpretation of the situation being modeled. When a conflict situation exists in a simulation, the user must decide which policy is the best interpretation for that situation. In the filling example shown earlier we used *Priority on event time*, which is the default policy used in this software.

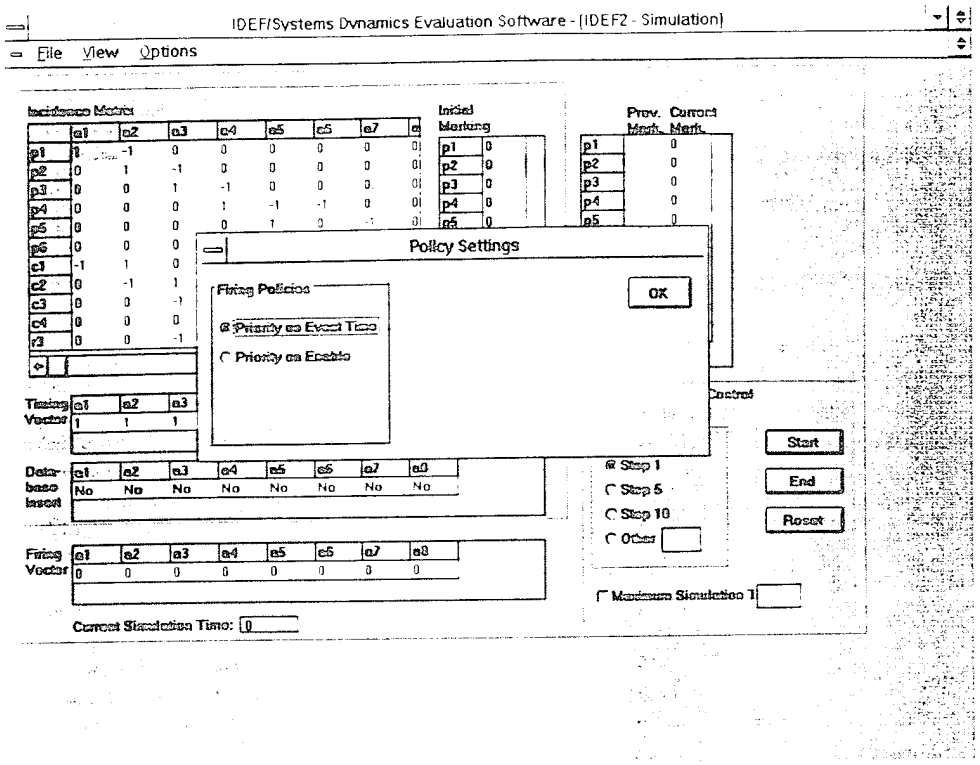


Figure 39 Policy Option Submenu

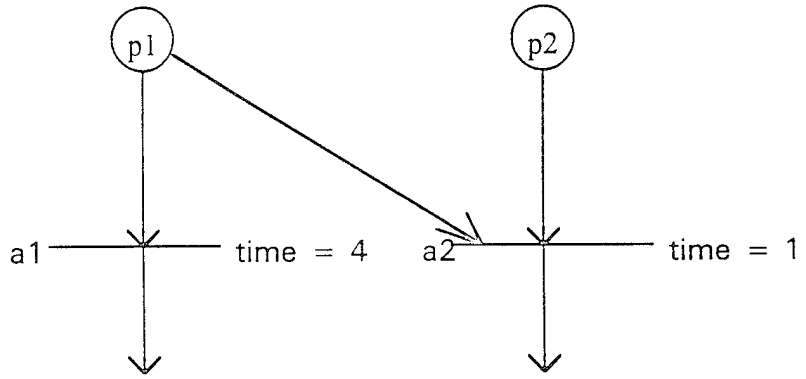


Figure 40 Example of a Conflict Requiring a Firing Policy

5.2 Petri Net Invariants

The IDEF2 - Simulation model is based on the Petri net modeling methodology, which is rich in mathematical features. Two fundamental mathematical concepts used are the T-invariants and P-invariants of the net. These provide tools that can be used to explore properties of the simulation model. In this section we will describe the nature of these invariants.

5.2.1 Mathematical Modeling of Petri Net Dynamics

We have previously stated that the dynamic behavior of the simulation is defined by changes in its marking. The marking changes when a transition fires. A transition fires after its input states are marked. More formally, a transition tj is enabled in marking M if $M(pi) \geq I(pi\ tj)$.

When a transition tj fires it results in a new marking, M', which occurs by removing I(pi tj) tokens from each of its input places and adding O(pi tj) tokens to each of its output places. More formally, M' is reachable from M according to the equation:

$$M'(pi) = M(pi) + O(pi\ tj) - I(pi\ tj) \qquad (1)$$

So, for example, Figure 41 shows a change in state from M_{k-1} to M_k by the firing of t1.

Let u be a firing vector, where $u^T = \{ u(t1), u(t2), u(t3) \}$. Then,

$$M_k = M_{k-1} + Ou_k - Iu_k$$

$$M_k \quad M_{k-1} \quad O - I \quad u$$

$$\begin{vmatrix} 0 \\ 1 \\ 1 \\ 0 \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{vmatrix} \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix}$$

where a 1 in the firing vector indicates a firing of that transition. The matrix O-I is an nxm matrix referred to as the incidence matrix, A. It defines the topology of the bipartite graph. The columns of A indicate the input places (-1) and output places (1) of each transition.

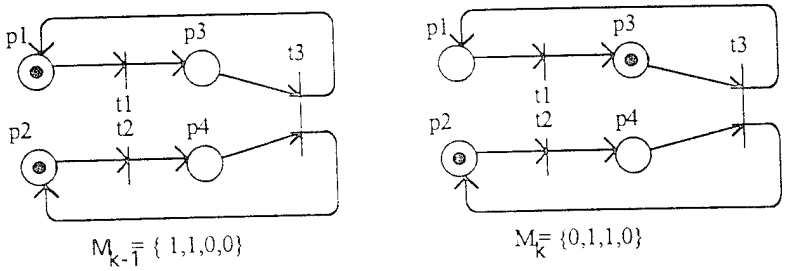


Figure 41 Change in Marking $M_{K-1} \rightarrow M_K$

One can imagine a sequence of firings given by $u_1 + u_2 + u_3 + \ldots$ Therefore, to arrive at some destination marking, M_d, from and initial marking, M_o:

$$M_d = M_0 + A \sum_{k=1}^{d} u_k$$

Let $\sum_{k=1}^{d} u_k = y$, $M_d - M_0 = \Delta M$. Then

$$Ay = \Delta M, \qquad (2)$$

and y is called the firing count vector. It is a vector whose elements are the number of times each transition fires in going from M_0 to M_d. Note that the firing count vector only shows the number of times that each transition is fired in the sequence; it does not uniquely identify the sequence in which the firing takes place.

5.2.2 Petri Net Invariants

A number of important properties of a PN can be evaluated using the concept of PN invariants. An invariant of a PN depends on its topology. There are two invariants of interest, the P-invariant and the T-invariant.

Definition: If there exists a set of non-negative integers, x, such that $x^T A = 0$, then x is called a P-invariant of the PN.

Let x be a weighting vector $x = [\, w_1 \;\; w_2 \;\; w_3 \ldots w_n \,]$. For the PN of Figure 41 we make the computation

$$[\, w_1 \; w_2 \; w_3 \; w_4 \,] \begin{vmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{vmatrix} = [\, 0 \;\; 0 \;\; 0 \,], \qquad (3)$$

$$\begin{aligned} -w_1 + \quad w_3 \quad &= 0 \\ -w_2 \quad + w_4 &= 0 \\ w_1 + w_2 - w_3 - w_4 &= 0, \end{aligned}$$

which has the following solutions:

$$\begin{array}{rcccc} & p1 & p2 & p3 & p4 \\ x_1 = (& 1 & 0 & 1 & 0) \\ x_2 = (& 0 & 1 & 0 & 1) \\ x_3 = (& 1 & 1 & 1 & 1) \end{array}$$

Although the above three solutions exist, we are only interested in the *minimal* set of P-invariants. If an invariant is covered by two or more other invariants, it is not in the minimal set. Therefore x_1 and x_2 are minimal P-invariants and they cover x_3. In general, there are (n-r) minimal P-invariants, where n is the number of places and r = the rank of A.

The meaning of a P-invarient can be seen by observing Figure 41. If one or more tokens exist in places p1 and/or p3, these two places will share the circulation of those tokens. This is disclosed by the invariant x_1. Similarly, places p2 and p4 may share the circulation of tokens. This is disclosed by the invariant x_2. The minimal P-invariants disclose the minimal structure of circuits that will share tokens. This also inplies that these tokens are *conserved* within the circuit; i.e., the number of tokens in the set of places which make up the circuit, when weighted by the vector x, is a constant.

Definition: A vector,y, of non negative integers is a T-invariant iff there exists a marking M and a firing sequence back to M whose firing count vector is y.

Since Ay = ΔM as given in Eq.(2), a T-invariant is a solution to the equation Ay = 0. Let y be a vector $y = [\, u_1 \;\; u_2 \;\; \ldots \;\; u_m \,]$. For the PN of Figure 41,

$$\begin{vmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}$$

$$\begin{aligned} -u_1 \quad + u_3 &= 0 \\ -u_2 + u_3 &= 0 \\ u_1 \quad - u_3 &= 0 \\ u_2 - u_3 &= 0 , \end{aligned}$$

which yields the T-invariant $y^T = (\, 1 \;\; 1 \;\; 1 \,)$. The meaning of the T-invariant is clear from Eq. (2). Since $\Delta M = 0$, y defines the number of times each transition must fire in one complete cycle from M_0 back to itself. In this instance, each transition will fire once during a cycle. In general, there are (m-r) T-invariants, where m is the number of transitions.

The P and T-invarants of a PN provide tools to explore important properties of the system being simulated by the PN. In the following sections we will examine some properties of a PN.

5.2.3 Reachability

It is often important to know what states a system can reach from some initial state. The set of all markings that can occur from some intial marking due to the firing of transitions defines the reachable states of the PN.

Property: If a P-invariant exists, then any reachable marking, M', from a marking M_0, must satisfy the relation $x^T M' = x^T M_0$.

Proof: $M' = M_0 + Au$

$x^T M' = x^T M_0 + x^T Au$

since $x^T A = 0$,

$x^T M' = x^T M_0$ (4)

EXAMPLE

With reference to Figure 41 ,show that M=(0 0 1 1) is reachable from M_0 but M=(1 0 1 0) is not reachable from M_0.

Answer

Any reachable marking must satisfy Eq. (7.4), where M_0 = (1 1 0 0).

Testing M = (0 0 1 1):

$$(1\ 0\ 1\ 0) \begin{vmatrix} 0 \\ 0 \\ 1 \\ 1 \end{vmatrix} = (1\ 0\ 1\ 0) \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} \qquad (0\ 1\ 0\ 1) \begin{vmatrix} 0 \\ 0 \\ 1 \\ 1 \end{vmatrix} = (0\ 1\ 0\ 1) \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix}$$

$$1 = 1 \qquad\qquad 1 = 1$$

Testing M = (1 0 1 0):

$$(1\ 0\ 1\ 0)\begin{vmatrix}1\\0\\1\\0\end{vmatrix} \neq (1\ 0\ 1\ 0)\begin{vmatrix}1\\1\\0\\0\end{vmatrix}$$

$$2 \neq 1$$

The reachable markings can also be found by firing transitions until all the states of the system have been reached. This yields the *reachability graph* of the net, as shown in Figure 42. This procedure for identifying reachable states is suitable for small problems, but can be unsatisfactory for large problems because the state space explodes.

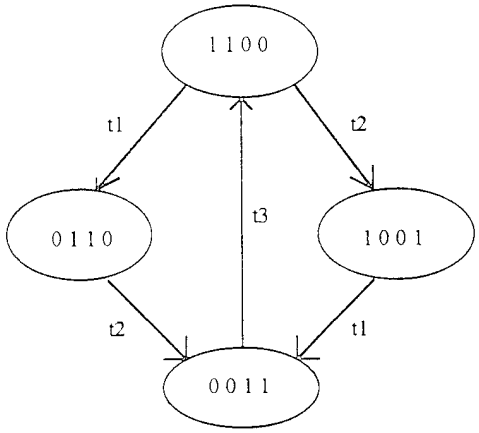


Figure 42 Reachability Graph for PN of Figure 41

5.2.4 Boundedness

A typical problem in manufacturing is to determine required buffer capacities between machining or assembly operations within the system in order to handle work in process. If tokens are used to represent parts being manufactured and places are used to represent buffers, it is of interest to know that the buffer capacities will not be exceeded. A PN is said to be k bounded if the number of tokens in each place does not exceed a finite number k for every reachable marking from M_0. A PN is said to be safe if it is 1 bounded. For example, the PN of Figure 41is safe and the PN of Figure43 is unbounded.

In the above definition, boundedness depends on the initial marking. A stronger condition for boundedness is structural boundedness, which means that the PN is bounded for any finite marking M_0.

Property: A Petri net is structurally bounded iff there exists a non zero vector, x, of non negative integers such that $x^T A \leq 0$.

Proof: From Eq.(2), $M = M_0 + Ay,\ y \geq 0$.
Therefore, $x^T M = x^T M_0 + x^T Ay$.
Since $x^T A \leq 0$ and $y \geq 0$, $x^T Ay \leq 0$ and $x^T M \leq x^T M_0$
Therefore, for any marking, the number of tokens in all marked places has an upper bound as follows:

$$\sum_p x_p{}^T M \leq x^T M_0$$

Therefore, an upper bound for any individual place, p, is:

$$x(p)\, M(p) \leq x^T M_0$$

$$M(p) \leq (M_0{}^T x) / x(p),$$

where x(p) is the pth entry of x.

Another way to look at structural boundedness is to observe that a net becomes unbounded when there exists the condition that, when its transitions fire, more tokens are added to output places than are absorbed from input places. Therefore, a necessary and sufficient condition for boundedness is that $x^T O \leq x^T I$, which yields the equation $x^T A \leq 0$.

Figure 43 is not structurally bounded. This is obvious since, for every firing of t1, tokens are sent to p2 as well as p3. Eventually, tokens will accumulate in the lower circuit without bound. Applying the criteria above,

$$x^T A \leq 0$$

$$\begin{vmatrix} w_1 & w_2 & w_3 & w_4 \end{vmatrix} \begin{vmatrix} -1 & 0 & 1 \\ 1 & -1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{vmatrix} \leq \begin{vmatrix} 0 & 0 & 0 \end{vmatrix}$$

which yields the inequalities:

$$\begin{aligned} -w_1 + w_2 + w_3 &\leq 0 \\ -w_2 + w_4 &\leq 0 \\ w_1 + w_2 - w_3 - w_4 &\leq 0 \end{aligned}$$

The fact that there is no solution to the system of inequalities indicates the net is structurally unbounded for any initial marking. It should be pointed out that a structurally unbounded net may not be unbounded for a particular initial marking. Structural unboundedness refers to the fact that the topology of the net is such that there is at least one initial marking for which the net will be unbounded.

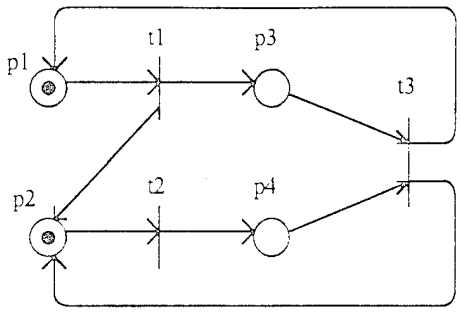


Figure 43 Unbounded Petri Net

In Section 5.2.2 we discussed the fact that the P-invariant defined a circuit of the net in which tokens were conserved; i.e., the number of tokens in the places of the circuit are constant when weighted by the vector x. By implication, if there exists a set of P-invariants that cover all places in PN, the weighted number of tokens in PN are constant. This would also imply that the net is bounded. Hence, another test of boundedness is to test for the conservative property; i.e., $x^T A = 0$, where x is a positive integer, $x \neq 0$.

5.2.5 Summary on Invariants

The use of invariants as a tool for analyzing a system model has been demonstrated. In this software we have given the user a facility for computing the invariants after the simulation model is generated. The selection items for the T-invariants and the P-invariants can be found under the *Analysis* menu item. It is left to the user to enter the model of Figure 41 and to compute the invariants and to compare them with the results shown in 5.2.2.

5.3 Generating a Discrete Controller Program

IDEF/System Dynamics software can also be used to illustrate how an IDEF0 / Petri net incidence matrix structure can be used to generate discrete control logic. In the following sections we will introduce an illustrative example and guide the user through the process of entering the model.

5.3.1 A Filling Line Example

Consider the situation shown in Figure 44. The activities along this production line are controlled by a supervisory controller. We are interested in modeling the problem and the controls from the point-of-view of the supervisory controller.

The package is moving along a conveyor until it reaches sensor 1. The conveyor is stopped momentarily by the supervisory controller while the filler dispenses product into the conveyor. When complete, the filler sends a "filling done" signal to the supervisor and the conveyor is restarted. The package passes to the weighing station, where the conveyor is momentarily stopped while the package is weighed. The scale provides two signals to the supervisor, a "weighing done" signal and a "reject" or "accept" signal. Based on these signals the package is either sent through the accept lane or diverted down the reject lane. For simplicity we assume that only one package is allowed through this portion of the line at a time. Continuous operation could also be assumed, but it requires a different incidence matrix than the one that will be introduced in this discussion.

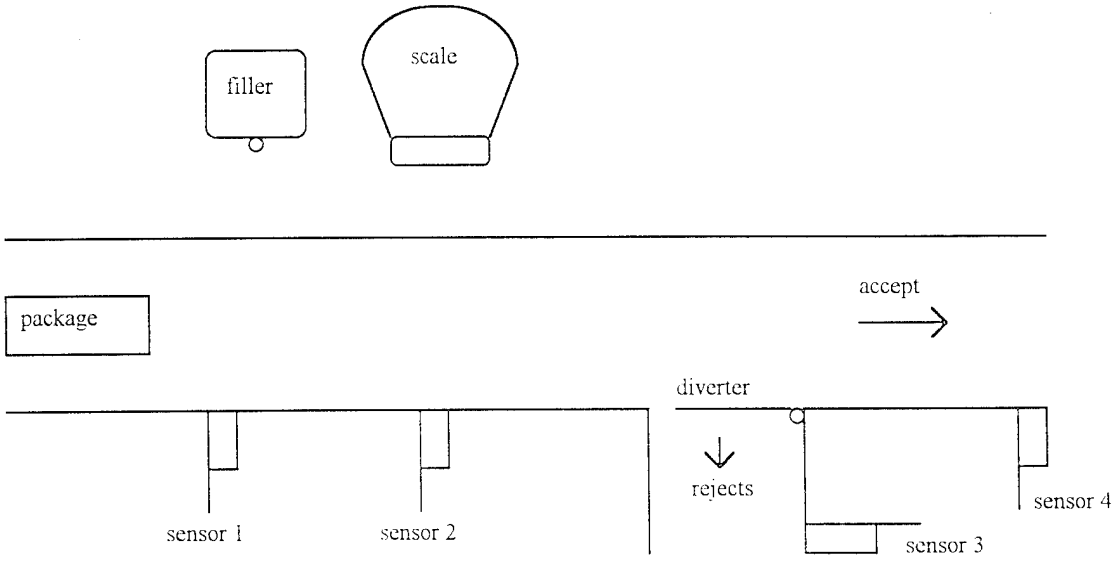


Figure 44 Filling Line Example Situation

A Petri net model that represents this situation is shown in Figure 45. The packages exiting at p5 and p6 give control back to the front of the line for the reentry of another package. This is indicated by the arcs reentering at p1. Thus, the model is showing the control logic.

The transitions between states occur as a result of the input signals from the sensors along the line. The transitions of Figure 45 are labeled with the appropriate signal. The activities along the line must actuate the outputs to control the overall operation of the line. The approriate output for an activity is shown next to the activity. It is left to the reader to examine the control model and confirm that it is appropriate to the situation.

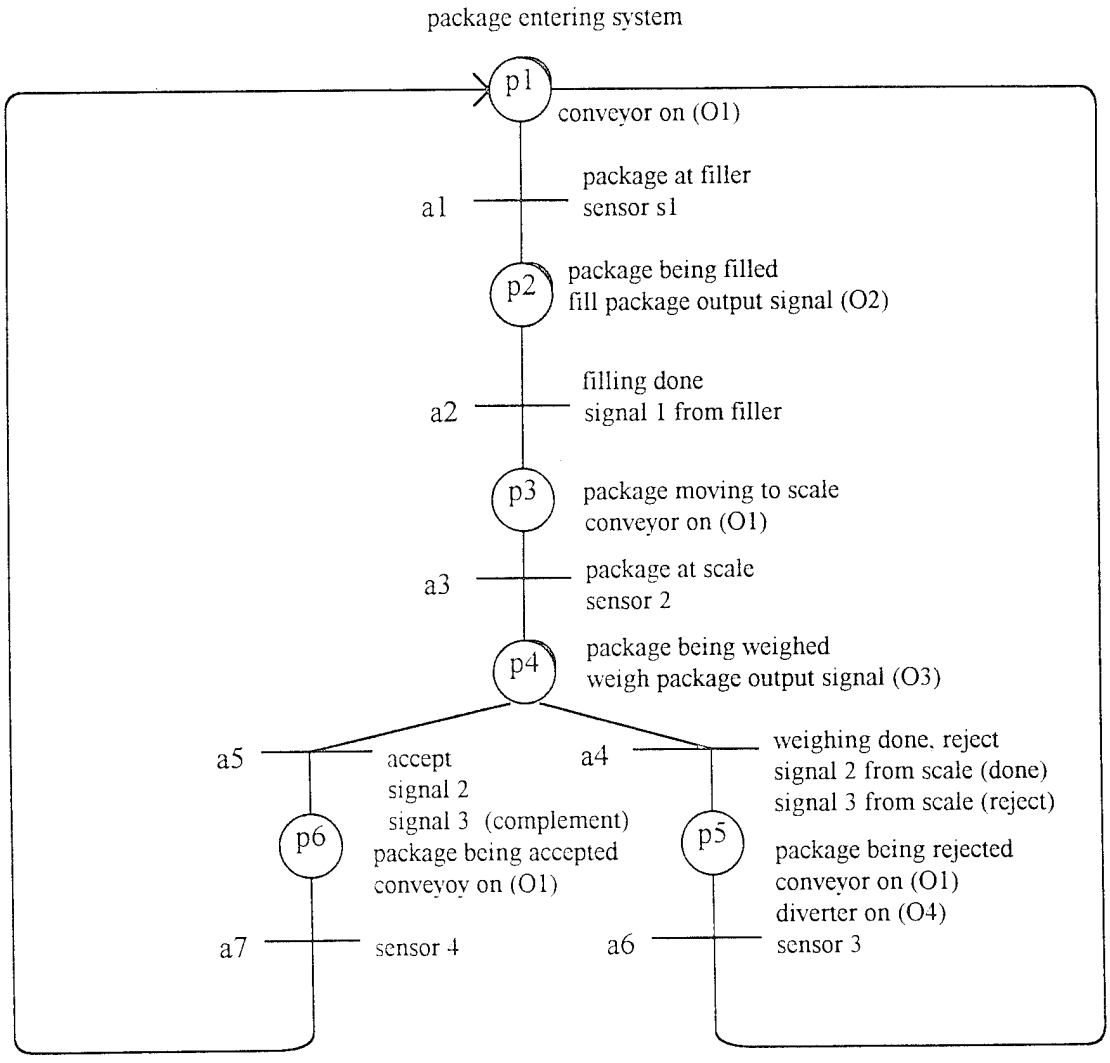


Figure 45 Petri Net of Filling Line Example

5.3.2 Entering Problem Data

The incidence matrix can be entered in the usual way, as previously shown with other example problems. The incidence matrix for this example is as follows:

	*a*1	*a*2	*a*3	*a*4	*a*5	*a*6	*a*7
*p*1	−1					1	1
*p*2	1	−1					
*p*3		1	−1				
*p*4			1	−1	−1		
*p*5				1		−1	
*p*6					1		−1

The reader may enter the incidence matrix data or load the problem from the IDEF directory, where the project name is "fill.idf". In using the incidence screen for modeling controller problems, it is not necessary to enter mechanism data. It is assumed that all inputs and outputs are going to be entered with respect to the supervisory controller.

Once the model is in active memory, the user should exit the incidence matrix screen and click on the *Edit* menu item and the *Input/Output* submenu item. The user will be presented with a screen having the format of Figure 46. The input box is where the user enters the signaling and sensor information. The output box is where the user enters the supervisory controller output actions. A description of the inputs and outputs can be entered in the descriptions panels. If the user is entering the data, the appropriate input and output matrices are given below. Note that the entry "2" indicates the complement state of I5. That is, if I5 enabled is "1", I5 disabled is "2". Since I4 and I5 are the input conditions for both a4 and a5, it is necessary to be specific about the state of I5. ***The user must click on a cell and type the entry into the cell using the keyboard.***

	*I*1	*I*2	*I*3	*I*4	*I*5	*I*6	*I*7
*a*1	1						
*a*2		1					
*a*3			1				
*a*4				1	1		
*a*5				1	2		
*a*6						1	
*a*7							1

	*O*1	*O*2	*O*3	*O*4
*p*1	1			
*p*2		1		
*p*3	1			
*p*4			1	
*p*5	1			1
*p*6	1			

Exit the Input/Ouput Screen and return to the project screen. Click on the menu item *Make* and the submenu item *Ladder Logic*, as shown in Figure 47. The software will compute the control logic and represent it in ladder logic form. There will be multiple

pages of the logic code shown on the screen. The user can review earlier pages by minimizing or closing each page. The logic underlying this ladder is created in a particular manner. The interested reader is referred to reference [1].

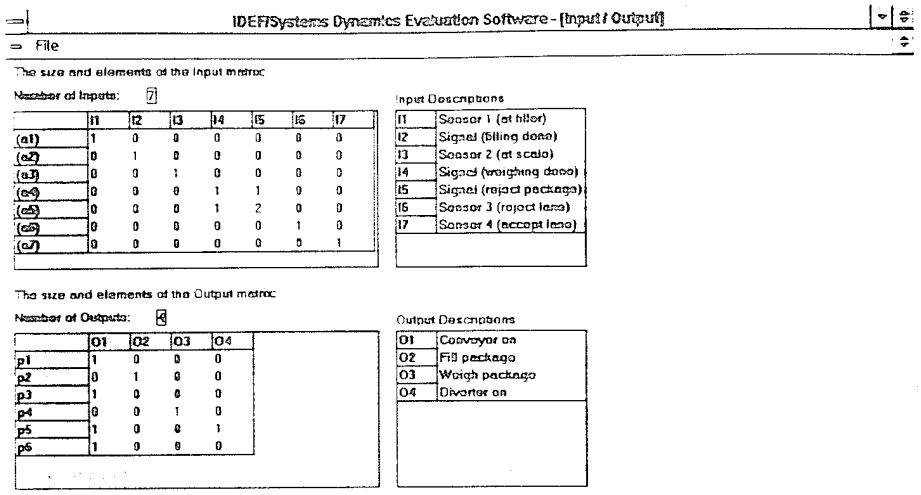


Figure 46 Input/Output Data Entry

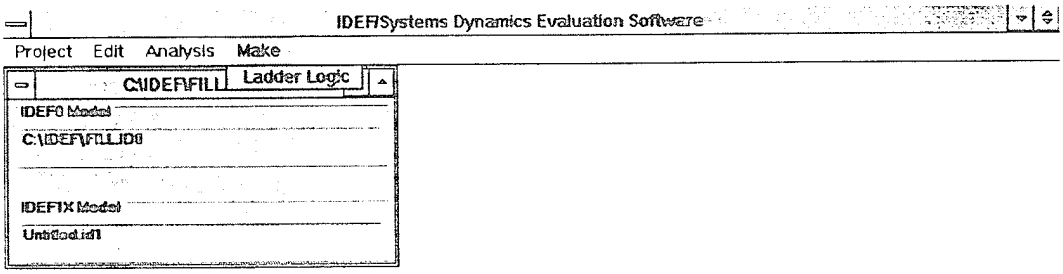


Figure 47 Selection of Ladder Logic Submenu

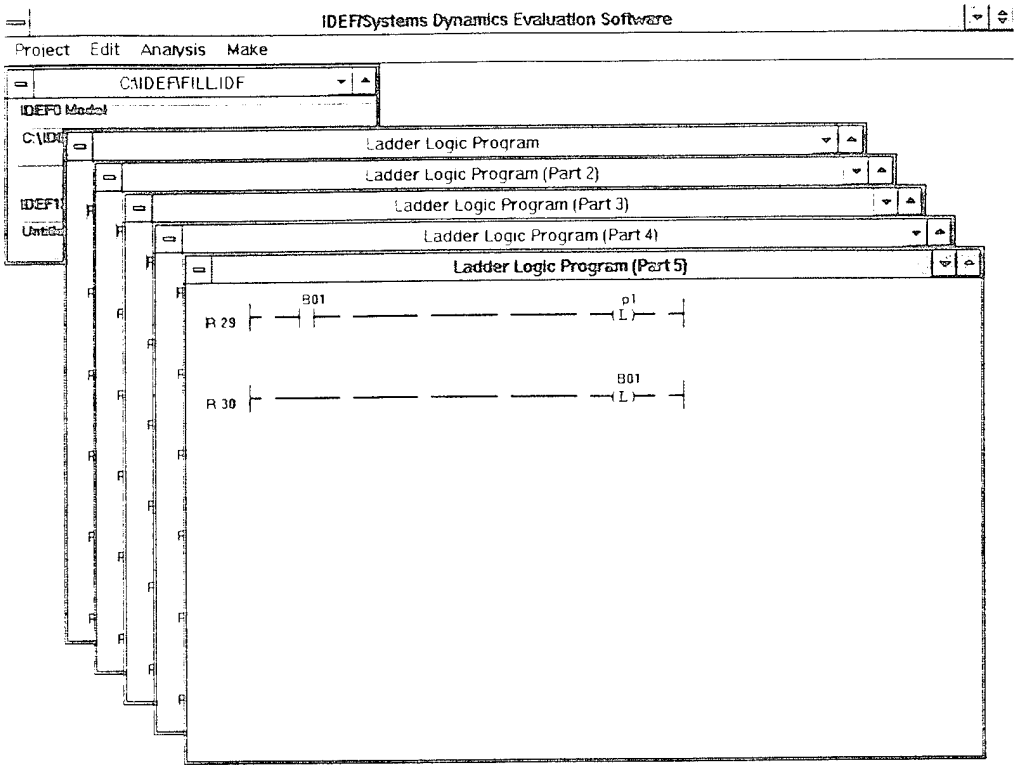


Figure 48 Ladder Logic Output

5.3.3 Summary on Generating Discrete Control Logic

The use of the incidence matrix structure to represent both an IDEF0 model and a Petri net has been shown. Using an incidence matrix in conjunction with input/output information, it is demonstrated how discrete controller logic can be defined. This prototype development is currently limited to small problem sizes (8 inputs and 8 outputs) and is used for problems which can be defined as a cyclic model, such as the example shown in this section.

REFERENCES

[1] Boucher, T.O. (1996) *Computer Automation in Manufacturing*, Chapman & Hall, London.

[2] Boucher, T.O. and M.A. Jafari (1992) Design of a factory floor sequence controller from a high level system specification, *Journal of Manufacturing Systems*, Vol. 11, No. 6.

[3] Desrochers, A.A. and Al-Jaar, R.Y. (1995) *Applications of Petri Nets in Manufacturing Systems: Modeling, Control and Performance Analysis*, IEEE Press, Piscataway, New Jersey.

[4] Jafari, M.A. and T.O. Boucher (1994) A rule based system for generating a ladder logic control program from a high level systems model, *Journal of Intelligent Manufacturing*, Vol. 5, No. 1.

[5] Murata, T. (1989) Petri nets: properties, analysis and applications, *Proceedings of IEEE*, Vol. 77, No. 4.

[6] Softech, Inc. (1981) Integrated Computer-Aided Manufacturing (ICAM) Final Report: *IDEF0 Functional Modeling Manual*, Contract No. F33612-78-C-5158.

REFERENCES

[1] Boucher, T.O. (1996) *Computer Automation in Manufacturing*, Chapman & Hall, London.

[2] Boucher, T.O. and M.A. Jafari (1992) Design of a factory floor sequence controller from a high level system specification, *Journal of Manufacturing Systems*, Vol. 11, No. 6.

[3] Desrochers, A.A. and Al-Jaar, R.Y. (1995) *Applications of Petri Nets in Manufacturing Systems: Modeling, Control and Performance Analysis*, IEEE Press, Piscataway, New Jersey.

[4] Jafari, M.A. and T.O. Boucher (1994) A rule based system for generating a ladder logic control program from a high level systems model, *Journal of Intelligent Mamufacturing*, Vol. 5, No. 1.

[5] Murata, T. (1989) Petri nets: properties, analysis and applications, *Proceedings of IEEE*, Vol. 77, No. 4.

[6] Softech, Inc. (1981) Integrated Computer-Aided Manufacturing (ICAM) Final Report: *IDEF0 Functional Modeling Manual*, Contract No. F33612-78-C-5158.